TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TP.HCM
KHOA CÔNG NGHỆ THÔNG TIN

CẤU TRÚC DỮ LIỆU & GIẢI THUẬT
BÀI TẬP PROJECT P2

Giảng viên: Nguyễn Tri Tuấn

MỤC LỤC

I. Thông tin cá nhân

- Họ và tên: Nguyễn Văn Nhật
- MSSV: 1612457
- Lớp: 16CTT3
- Email: vannhat8198@gmail.com
- SĐT: 0976827921

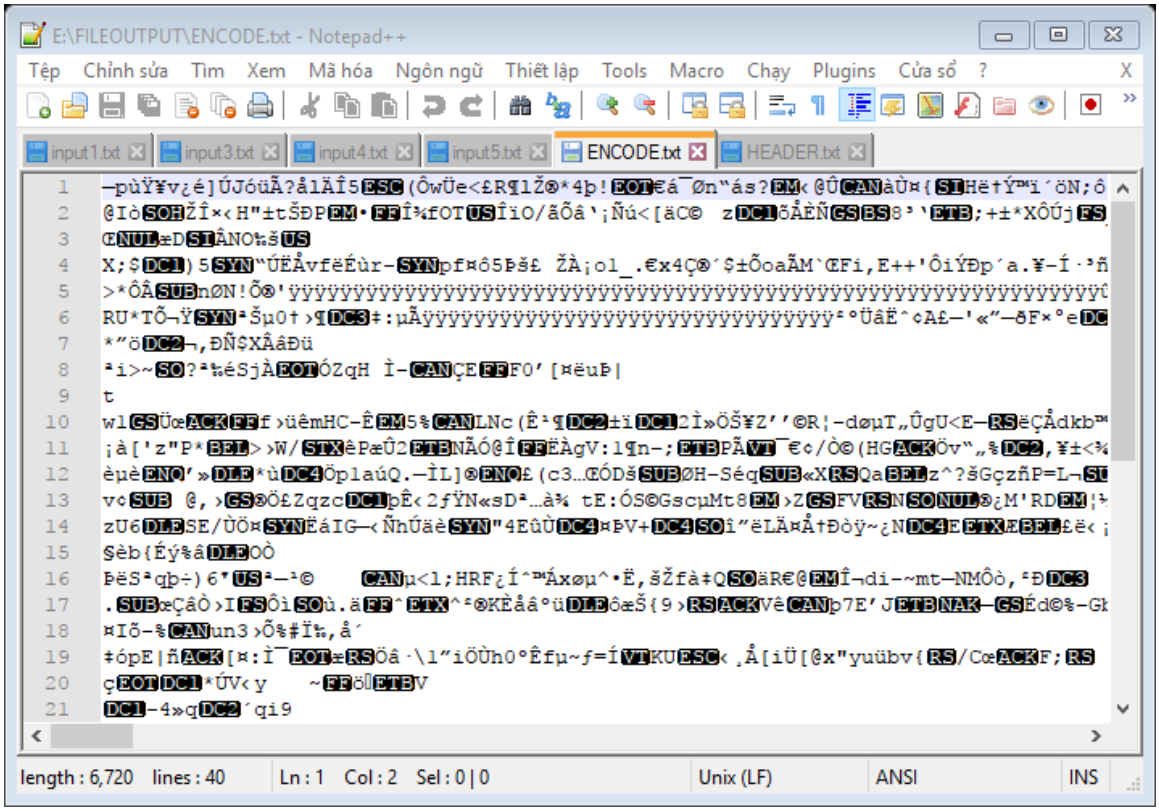
- Source code được viết bằng Visual Studio Professional 2015

II. Báo cáo bài làm

1. Cấu trúc file nén

File nén bao gồm 2 file chính:

+ File Encode chứa nội dung của toàn bộ các file trong thư mục được nén. File Encode được ghi dưới dạng nhị phân gồm các byte nối liền nhau.



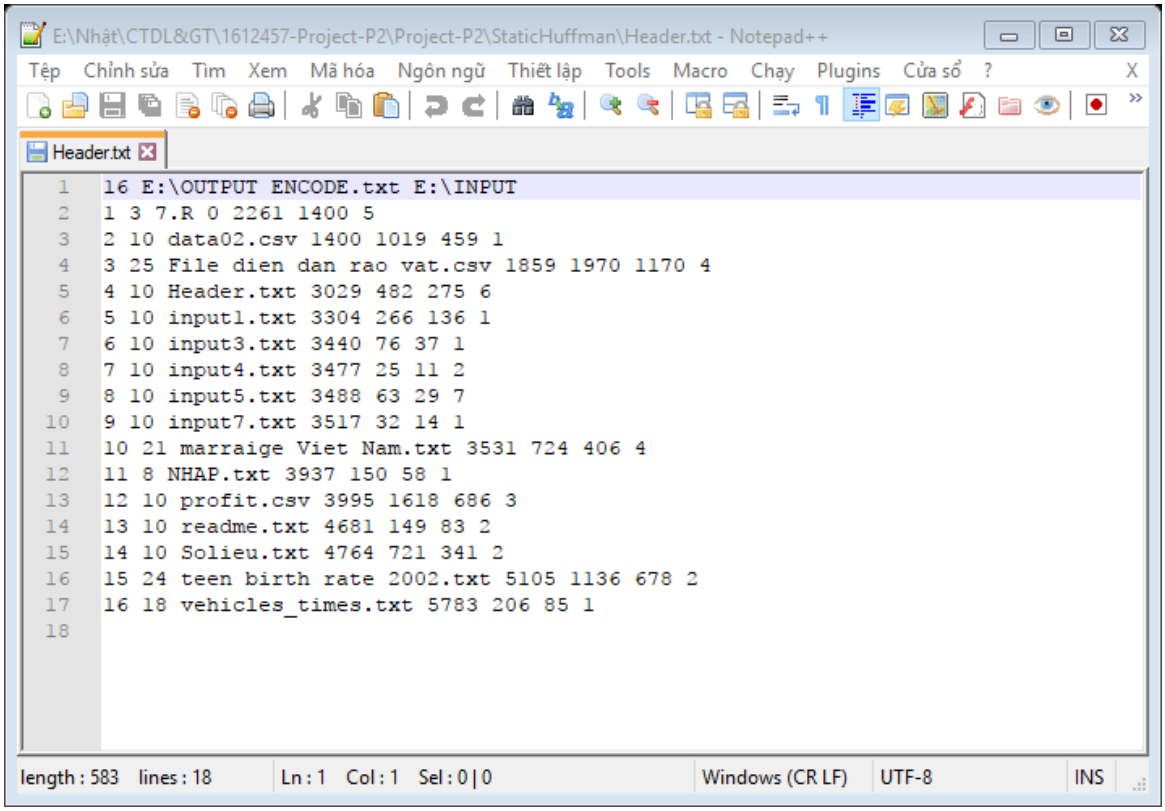
+ File Header chứa thông tin của từng file trong thư mục đã được nén, file Header sẽ được lưu trong thư mục lưu project và có cấu trúc như sau:

- Dòng thứ nhất là gồm 4 trường thông tin theo thứ tự như sau:

  Số file trong thư mục đã được nén|Đường dẫn tới thư mục chứa file nén (*file_out*)|Tên file nén (*file_out)*|Đường dẫn tới thư mục chứa các file cần nén (*folder_in)*

- n dòng tiếp theo, mỗi dòng chứa thông tin của từng file, gồm 7 trường thông tin theo thứ tự như sau:

STT|Độ dài tên file|Tên file|Vị trí bắt đầu của nội dung đã được nén trong file nén (*file_out)*|Kích thước trước khi nén|Kích thước sau khi nén|Số bit có nghĩa của byte cuối



- File nén được tách ra thành 2 file Encode và Header sẽ thuận tiện cho quá trình giải nén. Cấu trúc file Header được thiết kế bao gồm các thông tin như trên khi đọc file Header người dùng có thể dễ dàng tìm được vị trí nội dung cần giải nén trong file Encode và giải nén một cách nhanh chóng hiệu quả. Quá trình ghi thông tin vào file Header được thực hiện khi nén xong một file và quá trình ghi kết thúc khi nén xong tất cả các file trong thư mục.

2. Các cấu trúc dữ liệu đã dùng

Dach sách liên kết đơn (Linked List): được sử dụng để lưu các tên file dạng chuỗi được lấy ra trong thư mục cần nén. DSLK đơn sử dụng trong chương trình này gồm 2 con trỏ Head và Tail, bao gồm thao tác chính là thêm vào cuối danh sách (AddTail).

- Cài đặt chính như sau:

```
void AddTail(Node*&head, Node*&tail, char* x){
	Node*p = getNode(x);
	if (head == NULL)head = tail = p;
	else {
		tail->next = p;
		tail = p;
	}
}
void LayTenFile(Node*&head, Node*&tail, char*DuongDan){
	std::experimental::filesystem::path p(DuongDan);
	for(auto i=directory_iterator(p);i!=directory_iterator();i++)
	{
		if (!is_directory(i->path())){
			string x;
			x = i->path().filename().string();
			char *tmp = new char[x.length() + 1];
			tmp = std::strcpy(tmp, x.c_str());
			AddTail(head, tail, tmp);
	// lấy tên file đưa vào cuối dslk đơn
		}
		else
			continue;
	}
}
```

Ngoài ra còn sử dụng thêm các cấu trúc để tạo cây Huffman với các thuật toán tương ứng.

3. Các hàm chính trong chương trình

a. Cấu trúc 1 node của cây Huffman và cấu trúc mã bit 1 kí tự được cài đặt như sau:

```
struct HUFFNODE {
     unsigned char  c; // ký tự
     int freq; // tần số
     bool used; // đánh dấu
node đã xử lý hay chưa
     int nLeft; // chỉ số node
con trái
     int nRight; // chỉ số node
con phải
};
```

```
struct MABIT {
     char*bits; // mảng tạm lưu
các bit của 1 kí tự
     int soBit; // số bit của
kí tự đó
};
```

b. Chương trình chính được cài đặt sử dụng một số biến toàn cục như sau:
   - `HUFFNODE huffTree[MAX_NODE]:` mảng chứa các node của cây Huffman. MAX_NODE = 511.
   - `MABIT bangMaBit[256]:` Mảng chứa mã bit của tối đa 256 kí tự trong cây Huffman. Chỉ số mảng bằng với mã ASCII của kí tự.
   - `int posRoot:` Chỉ số của node gốc trong câu Huffman.
   - `HUFFNODE ROOT`: node gốc của cây Huffman.

c. Một số hàm chính:

Hàm khởi tạo: `void KhoiTao();` sẽ khởi tạo toàn bộ các node trong mảng huffTree mỗi node chứa 1 kí tự có mã ASCII bằng chỉ số mảng, khởi tạo tần số xuất hiện mỗi kí tự bằng 0, chỉ số con trái và phải bằng -1, đánh dấu các node chưa sử dụng used = false.

```
for (int i = 0; i < MAX_NODE; i++) {
          huffTree[i].c = i;
          huffTree[i].freq = 0;
          huffTree[i].used = false;
          huffTree[i].nLeft = -1;
          huffTree[i].nRight = -1;
     }
```

Hàm thống kê tần số xuất hiện của các kí tự:

- Tên hàm và prototype:
  `void ThongKeTanSoXuatHien(char* tenFile);`
- Input: Một chuỗi kí tự **char*** chứa file cần duyệt để tạo bảng thống kê.
- Output: Hàm **void** nên không có giá trị trả về.

- Ý nghĩa của hàm: Hàm sẽ duyệt từ đầu đến cuối file và thống kê tần số xuất hiện của từng kí tự trong file đưa vào mảng toàn cục **huffTree** (tối đa là 256 node đầu tiên của mảng) khi kết thúc.
- Mô tả ngắn gọn ý tưởng thuật toán: Dùng một con trỏ đọc từng kí tự trong file, mỗi kí tự đọc được sẽ đưa vào mảng chứa các node tại vị trí bằng với mã ASCII của kí tự đó và tăng tần số xuất hiện lên 1. Tiến hành đến cuối file.

```
while (1) {
          fscanf(fi, "%c", &c);
          if (feof(fi))break;
          huffTree[c].freq++;
     }
```

Hàm tạo cây Huffman:

- Tên hàm và prototype:
  `int TaoCayHuffman();`
- Input: Không có tham số vào, hàm sử dụng mảng toàn cục **huffTree** để tạo cây Huffman.
- Output: Trả về chỉ số của node gốc sau khi tạo xong cây Huffman.
- Ý nghĩa của hàm: Hàm tạo ra cây Huffman hoàn chỉnh từ bảng thống kê tần số xuất hiện của các kí tự đã được lưu trong mảng **huffTree** và trả về chỉ số của node gốc (node có tần số lớn nhất trong mảng).
- Mô tả ngắn gọn ý tưởng thuật toán: Sử dụng hàm phụ là `bool Tim2PhanTuMin(int &i, int &j, int nNode)` để tìm ra chỉ số 2 node có tần số nhỏ nhất trong mảng **huffTree** là i và j, sau đó tạo mới 1 node cha (đánh dấu chưa sử dụng) với i là chỉ số node con trái, j là chỉ số node con phải. Đánh dấu 2 node có chỉ số i, j đã sử dụng (used = true) và đưa node cha vào mảng **huffTree**. Tiến hành quá trình tạo node cha để đưa vào mảng đến khi trong mảng **huffTree** chỉ còn 1 node được đánh dấu sử dụng và trả về chỉ số của node cuối đó cũng là chỉ số node gốc, khi đó cây Huffman đã được tọa ra lưu trong mảng **huffTree**.

```
    huffTree[nNode].c = huffTree[i].c;
    huffTree[nNode].freq = huffTree[i].freq +huffTree[j].freq;
    huffTree[nNode].nLeft = i;
    huffTree[nNode].nRight = j;
    huffTree[i].used = true;
    huffTree[j].used = true;
    huffTree[nNode].used = false;
```

Hàm phát sinh mã bit của từng kí tự:

- Tên hàm và prototype:
  `void PhatSinhMaBit(int nRoot);`
- Input: Chỉ số node gốc.
- Output: Hàm **void** nên không có giá trị trả về.
- Ý nghĩa của hàm: Hàm sẽ duyệt hết cây đến tất cả các kí tự có trong bảng thống kê tần số và đưa mã bit của mỗi kí tự vào mảng toàn cục **bangMaBit** dưới dạng chuỗi.
- Mô tả ngắn gọn ý tưởng thuật toán: Ban đầu khởi tạo tất cả các phần tử trong mảng toàn cục **bangMaBit** có chuỗi mã bit là NULL và số bit là 0. Sử dụng hàm phụ `void DuyetCayHuffman(int node, char maBit[], int nMaBit)` để tiến hành duyệt cây đến các node chứa kị tự cần mã hóa, nếu duyệt qua phải tiến hành thêm bit 1 vào bảng mã bit, ngược lại thêm bit 0. Tiến hành đệ quy để duyệt cây Huffman đến hết tất cả các node cần mã hóa thì dừng lại.

```
for (int i = 0; i < 256; i++) {
     bangMaBit[i].soBit = 0;
     bangMaBit[i].bits = NULL;
};
char maBit[MAX_BIT_LEN / 8];// MAX_BIT_LEN = 10000
int nMaBit = 0;
DuyetCayHuffman(nRoot, maBit, nMaBit);
```

Hàm mã hóa 1 kí tự:

- Tên hàm và prototype:
  `void MaHoa1KyTu(unsigned char c, unsigned char &out, unsigned char &viTriBit, FILE*f, int &count);`
- Input: Gồm các tham số **c**: là kí tự được mã hóa, **out**: kí tự sẽ được mã hóa đủ 8 bit và lưu và **File *f**, **viTriBit**: là vị trí của bit trong biến out sẽ bắt được bật và **count**: biến để lưu lại số byte sau khi nén.
- Output: Hàm **void** nên không có giá trị trả về.
- Ý nghĩa của hàm: Hàm truyền vào 1 kí tự **c**, dựa vào kí tự đó để lấy thông tin mã bit của kí tự được lưu tạm trong **bangMaBit** sau đó tiến hành bật các bit trong biến **out**, biến **out** đủ 8 bit sẽ được lưu vào file.
- Mô tả ngắn gọn ý tưởng thuật toán: Ban đầu dựa vào kí tự **c**, lấy bảng mã bit dựa vào mã ASCII của kí tự đó đã được lưu dưới dạng chuỗi trong mảng **bangMaBit** (gồm chuỗi bit và số bit). Tiến hành duyệt từ bit đầu đến bit cuối

của chuỗi mã bit, nếu là bit 1 thì tiến hành bật bit 1 của biến **out** giống với các bit trong chuỗi mã bit bắt đầu tại vị trí **viTriBit**, kiểm tra sau mỗi lần bật bit nếu **viTriBit** = 0 thì xuất biến **out** ra file, gán lại biến **out** = 0, **viTriBit** = 7 và biến đếm số byte sau khi nén **count** tăng 1, ngược lại giảm **viTriBit** xuống 1. Tiến hành cho đến khi hết chuỗi mã bit. Các tham số **out**, **viTriBit**, **count** được truyền tham chiếu vì quá trình mã hóa thực hiện nối bit liên tiếp các kí tự nên các giá trị này thay đổi và phải được lưu lại giá trị hiện tại để sử dụng cho lần mã hóa sau.

```
for (int i = 0; i < bangMaBit[c].soBit; i++) {
     if (bangMaBit[c].bits[i] == 1) {
          out = out | (1 << viTriBit); // bật bit
     }
     if (viTriBit == 0) {//du 1 byte, ghi byte do ra file
          viTriBit = 7;
          putc(out, f); out = 0; count++;
     }
     else viTriBit--;
}
```

Hàm nén:

- Tên hàm và prototype:
  `void NenHuffman(char*DuongDan, char*inputFile, FILE*Encode, FILE*Header, int&ViTriBatDau, int&STT);`
- Input: Gồm các biến **DuongDan** là đường dẫn tới folder cần nén, **inputFile** là tên file cần nén, **Encode** và **Header** là 2 file nén, **ViTriBatDau** là vị trí bắt đầu của nội dung file **inputFile** trong file **Encode**, **STT** là thứ tự file được nén.
- Output: Hàm **void** nên không có giá trị trả về.
- Ý nghĩa của hàm: Hàm sẽ duyệt tất cả các kí tự từ đầu đến cuối file và dựa vào mã bit của từng kí tự để nén thành từng byte (khi đủ 1 byte) một dưới dạng nhị phân vào trong file **Encode** và file **Header**.
- Mô tả ngắn gọn ý tưởng thuật toán: Dựa vào đường dẫn và tên file input, tiến hành mở file để lập bảng thống kê và tạo mã bit cho các kí tự trong file, tiến hành đọc lần lượt từng kí tự trong file sau đó sử dụng hàm **MaHoa1KiTu** để mã hóa và ghi byte xuống file. Đối với byte cuối cùng cần xác định được số bit có nghĩa của byte cuối dựa vào biến **viTriBit** từ hàm **MaHoa1KiTu** sau đó tiến hành lưu thông tin các file vào file **Header** sau khi tiến hành nén xong 1 file. Quá trình nén kết thúc khi đọc đến hết các kí tự trong file.

```
while (true) {
          fscanf(fi, "%c", &c);
          if (feof(fi)) {
               break;
          }
          MaHoa1KyTu(c, out, viTriBit,Encode,soByte);
     }
     soBitCoNghia = 7 - viTriBit; // Ban đầu viTriBit = 7
     if (soBitCoNghia == 0) {
          soBitCoNghia = 8;
     }
     else {
          putc(out, Encode);
          soByte++;
     }
               // ghi ra file header
     fprintf(Header, "%d %d %s %d %d %d %d\n",STT, DoDaiTenFile,
inputFile,ViTriBatDau, SoByteTruocKhiNen, soByte, soBitCoNghia);
```

Hàm giải nén:

- Tên hàm và prototype:
  `void DeCode (char *DuongDanFolderOutput, int ThuTuFileGiaiNen);`
- Input: Biến **char*** là đường dẫn đến folder chữa các file sau khi giải nén ra và biến **int** là thứ tự của file cần được giải nén.
- Output: Hàm **void** nên không có giá trị trả về.
- Ý nghĩa của hàm: Hàm sẽ đọc thông tin từ file **Header**, dựa vào thông tin đó tiến hành đọc tới vị trí của nội dung cần giải nén trong file **Encode** để giải nén file có thứ tự tương ứng.
- Mô tả ngắn gọn ý tưởng thuật toán: Ban đầu tiến hành đọc thông tin những file cần giải nén trong file **Header** (đọc dòng đầu để lấy thông tin file nén và dùng vòng lặp để đọc chính xác các dòng tiếp theo những thông tin của file cần giải nén đúng theo thứ tự file cần giải nén). Sau khi đọc xong file **Header**, dựa vào những thông tin đọc được, tiến hành mở lại file đó để tạo lại cây Huffman. Tiếp theo tiến hành đưa con trỏ đọc file tới vị trí của nội dung cần giải nén trong file nén dựa vào biến lưu vị trí bắt đầu được đọc từ file **Header** ở trên, đọc chính xác từng byte của toàn bộ nội dung giải nén và dùng một mảng kí tự tạm để lưu lại từng bit sau mỗi lần đọc. Tiến hành giải nén file

dựa vào cây Huffman vừa được tạo lại và chuỗi chứa bit, sau mỗi lần hoàn thành giải nén 1 file tiến hành kiểm tra checksum và thông báo kết quả.

Hàm lấy tên các file trong thư mục (không lấy tên thư mục con):

- Tên hàm và prototype:
```
void LayTenFile(Node*&head,Node*&tail,char*DuongDan);
```
- Input: 2 node **head** và **tail** là 2 con trỏ đầu và cuối của danh sách liên kết đơn, biến **char*** là đường dẫn tới thư mục cần nén.
- Output: Hàm **void** nên không có giá trị trả về.
- Ý nghĩa của hàm: Lấy tên tất cả các file trong thư mục được nén và đưa vào danh sách liên kết để lưu dưới dạng chuỗi (không lấy tên thư mục con).
- Mô tả ngắn gọn ý tưởng thuật toán: Sử dụng thư viện **filesystem** và lấy tham số đầu vào là biến **char*** lưu đường dẫn đến thư mục cần nén để tiến hành đọc toàn bộ tên file trong thư mục đó. Mỗi lần đọc tên 1 file sẽ tiến hành thêm vào cuối **AddTail** danh sách liên kết. Kết thúc khi đọc hết toàn bộ tên file trong thư mục đó.

(Source code mẫu ở mục II.2)

III. Nhận xét

Chương trình cơ bản hoàn thành được những yêu cầu đề bài như sau:

- Hoàn thành thuật toán nén Huffman để nén 1 file.
- Hoàn thành hàm lấy các tên file trong thư mục và nén tất cả các file.
- Kết hợp được nhiều cấu trúc dữ liệu, kết hợp nhiều thuật toán để giải quyết yêu cầu.
- Hoàn thành được các thao tác xử lí trên file và xử lí bit…
- Tuy nhiên, về phần giải nén vẫn còn gặp một số lỗi khi giải nén một số file đặc biệt…

Thuân lợi và những khó khăn:

- Thuận lợi mà bản thân nhận được sau bài Project P2 là những bài học kinh nghiệm về cách phối hợp các thuật toán, các cấu trúc dữ liệu, những bài học về cách xử lí file (file text, file nhị phân) và các thao tác xử lí bit (bật bit và lấy bit), kinh nghiệm về cách fix bug phát sinh trong quá trình xử lí chương trình, xử lí thuật toán… Đồng thời bản thân cũng có kinh nghiệm trong cách quản lí thời gian hoàn thành bài tập.
- Khó khăn bản thân gặp phải trong quá trình làm bài: khó khăn bước đầu về việc cài đặt xử lí thuật toán Huffman nên phải tìm hiểu nhiều nguồn tham khảo. Trong quá trình cài đặt chưa còn phát sinh nhiều lỗi không đáng có. Bản thân chưa chủ động làm bài trong thời gian đầu, một thời gian sau mới tiến hành làm bài tập nên thời gian bị hạn chế…

IV. Một số tài liệu tham khảo

Xử lí bit:
http://forums.congdongcviet.com/showthread.php?t=316

Thư viện **filesystem:**
http://en.cppreference.com/w/cpp/experimental/fs

Tham khảo từ Bài thực hành thuật toán nén Huffman trong môn học Thực hành Cấu trúc dữ liệu và Giải thuật.