ĐẠI HỌC QUỐC GIA TP HCM

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

--o0o--

BÁO CÁO ĐỒ ÁN P2

Môn: Cấu trúc dữ liệu và giải thuật

Họ tên: Phạm Hoàng Minh

MSSV: 1612380

Lớp: 16CTT3

Giảng viên phụ trách: thầy Nguyễn Tri Tuấn

Thời gian: 3 tuần

Yêu cầu: Viết chương trình nén/ giải nén file dùng thuật toán Huffman tĩnh

MỤC LỤC

I. CẤU TRÚC FILE NÉN

1. STRUCT

```
struct HUFFNode
{
	unsigned char c; //Kí tự
	long freq; //Tần số
	int left; //Node con trái
	int right; //Node con phải
};


struct TYPE_FILE
{
	string oldName; //Tên file trước khi nén
	unsigned int bytes; //Kích thước file trước khi nén
	unsigned int length; //Độ dài của dữ liệu ghi vào file nén
	short bit_LastByte; //Số bit không có nghĩa ở byte cuối
};
```

2. FILE NÉN

HEADER	char signature[8]	Mã nhận dạng file
	int num_of_file	Số lượng file
	long **freq	Tần số
	TYPE_FILE *header	Thông tin của file trước nén
DATA	File[1] File[2] File[3] …. File[num_of_file – 1] File[num_of file]	Dữ liệu từng file sau khi được mã hoá theo thứ tự

II. CÁC CẤU TRÚC DỮ LIỆU QUAN TRỌNG KHÁC

1. PRIORITY QUEUE

```
class PRIORITY_QUEUE
{
		struct KEY_TYPE
	{
		long freq; //Tần số
		unsigned char data; //Kí tự
		short address; //Địa chỉ
	};

	KEY_TYPE *key; //Mảng tĩnh chứa queue
	int rear;
	int maxSize; //Kích thước tối đa của queue

	void heapify(int i); //heapify ở vị trí "i"
public:
	PRIORITY_QUEUE(int size); //Khởi tạo heap với kích thức = “size”
	PRIORITY_QUEUE(const PRIORITY_QUEUE& aQueue);
	~PRIORITY_QUEUE(); //destructor

	//operator
	bool isEmpty();
	bool insert(KEY_TYPE item); //push
	bool deleteMin(KEY_TYPE &item); //pop
	void printQueue();
};
```

Priority queue dùng mảng tĩnh để xây dựng một cây min heap và điều chỉnh bằng thuật toán heapify.

2. QUEUE

```
class Queue
{
	struct  QueueNode
	{
		string data; //Lưu tên file
		QueueNode *next;
	};
	QueueNode *front;
	QueueNode *rear;
public:
	Queue(); //default constructor
	Queue(const Queue&); //copy constructor
	~Queue(); //destructor

	// operations
	bool isEmpty();
	bool enqueue(string newItem);
	bool dequeue(string &item);
	bool frontValue(string &item);
	bool PrintQueue();
	int size();
	int length();
};
```

Queue dùng danh sách liên kết đơn và dùng 2 con trỏ front – rear để quản lí.

III. MÔ TẢ CHƯƠNG TRÌNH

1. BIẾN TOÀN CỤC

```
HUFFNode **huffTree;
long **freq;
TYPE_FILE *header;
string** bit; //bảng bitcode
int* bytesAfter = NULL; //Size của file sau khi giải nén
```

Dùng mảng 2 chiều để lưu các thông tin cần thiết. Chiều thứ nhất là số lượng các file trong folder. Chiều thứ 2 là:

- freq, bit: chứa 256 phần tử tương ứng 256 kí tự ASCII.

- huffTree: chứa 511 phần tử tương ứng có tối đa 511 node.

2. CÁC HÀM CHÍNH

void get_all_files_names_within_folder(string folder, Queue& names);

Ý nghĩa: Dùng để duyệt các file trong 1 folder.

Thuật toán: Dùng loop và lệnh FindNextFile trong thư viện Windows.h để enqueue vào list names.

Input: Tên folder và queue chứa danh sách tên các file.

string inverse(string &str);

Ý nghĩa: nghịch đảo chuỗi truyền vào.

Output: chuỗi sau khi nghịch đảo.

bool Allocate(int size, bool isCompress);

Ý nghĩa: cấp phát vùng nhớ cho chương trình

Input: số lượng file và cách phân biệt giữa nén và giải nén.

Output: Kết quả của việc cấp phát.

void Free(int size, bool isCompress);

Ý nghĩa: giải phòng vùng nhớ đã cấp phát.

Input: số lượng file và cách phân biệt giữa nén và giải nén.

int createHuffTree(int num);

Ý nghĩa: Tạo ra một cây huffTree

Thuật toán: Dùng priority queue để enqueue 2 phần tử nhỏ nhất trong bảng tần số, tìm ra node cha rồi insert vào cây huffTree và priority queue. Thuật toán dùng loop cho đến khi queue rỗng là ta được một cây huffTree hoàn chỉnh.

Input: số thứ tự của file tương ứng trong folder.

Output: vị trí của nRoot.

//NÉN

bool readFileToEncode(string filename, string type, int pos);

Ý nghĩa: Đọc file lần thứ nhất, mục đích chỉ để lưu bảng tần số, kích thước file, tên file trước khi nén.

Input: đường dẫn đến file, tên file, số thứ tự của file trong folder

Output: kết quả của việc đọc file.

void processHuffTree(int nNode, string bit, int num);

Ý nghĩa: Hàm duyệt cây để tạo ra bảng bitcode .

Thuật toán: dùng đệ quy để gọi hàm lại nhiều lần cho đến khi đi đến node lá, bitcode sẽ được lưu vào chuỗi string bit.

Input: nRoot, chuỗi chứa bitcode, số thứ tự của file trong folder

void createBit(char &c, char bit, int i);

Ý nghĩa: chuyển một chuỗi 8 bit biểu diễn bằng bitcode sang kí tự tương ứng trong bảng ASCII.

void writeHeader(ofstream& fOutput, int& pos, int size);

Ý nghĩa: Ghi phần header vào trước, lưu vị trí con trỏ ghi đang ghi đến phần data được nén vào tham số pos.

Input: Tên file ghi, biến rỗng để lưu vị trí con trỏ, số lượng file.

bool encode(int size,string* fInput, string fOutput);

Ý nghĩa: Đọc file lần 2, vừa đọc vừa encode dữ liệu rồi chuyển bitcode sang kí tự và ghi vào file nén.

Input: số lượng file, danh sách file nén, tên file sau khi nén.

Output: Kết quả của việc encode.

//GIẢI NÉN

bool readFileToDecode(string filename, int& size, int& pos);

Ý nghĩa: Đọc phần header, kiểm tra mã nhận dạng, lưu vị trí con trỏ đang đọc đến phần data được nén vào tham số pos.

Input: Tên file cần giải nén, biến rỗng để chứa số lượng file, biến rỗng để chứa vị trí con trỏ.

Output: Kết quả của việc đọc file nén.

unsigned char reprocessingHuffTree(string bin, int &pos, int nNode, int num);

Ý nghĩa: Chuyển từ bitcode sang kí tự bằng cách duyệt lại cây.

Thuật toán: Dùng đệ quy để gọi lại hàm nhiều lần, khi duyệt tới node lá sẽ trả về kí tự tương ứng với node đó.

Input: Chuỗi chứa dãy nhị phân sau khi duyệt cây, vị trí tiếp theo thêm vào "1" hoặc "0" trong chuỗi bin, node đang duyệt, số thứ tự của file.

Output: Kí tự trong bảng ASCII sau khi duyệt đến node lá.

bool checkSum(int sizeBefore, int sizeAfter);

Ý nghĩa: Kiểm tra size trước khi nén và sau khi giải nén, nếu sai thì báo lỗi.

Input: size trước khi nén (header.bytes), size sau khi nén (bytesAfter).

Output: Nếu khác nhau sẽ trả về false.

bool decode(string fInput, string folder, string file, int& size);

Ý nghĩa: Vừa đọc, vừa decode, vừa ghi dữ liệu. Decode kí tự sang đoạn bitcode rồi duyệt cây huffTree trả về dữ liệu và ghi ra file.

Input: tên file nén, tên folder cần giải nén ra, tên những file cần giải nén, số lượng các file sau khi giải nén.

Output: Kết quả của việc decode.