

ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



BÁO CÁO PROJECT I
CÁC THUẬT TOÁN NÉN NGUỒN RỜI RẠC

Chủ đề 1: Thuật toán mã hóa của Huffman

GVHD: PGS.TS Đặng Văn Chuyết

Thành viên nhóm	MSSV	Lớp
Lê Minh Việt Anh (nhóm trưởng)	20210037	IT2-05-K66
Nguyễn Mạnh Chiến	20210118	IT2-03-K66
Nguyễn Trọng Nhật	20215625	IT2-05-K66
Lê Việt Quang	20215630	IT2-05-K66

Hà Nội, tháng 01 năm 2024

MỤC LỤC

1. Giới thiệu đề tài.....	3
2. Phân tích đề tài.....	4
2.1. Thuật toán mã hóa của Huffman.....	4
2.2. Phân tích yêu cầu đề tài	4
2.3. Phân công nhiệm vụ.....	4
3. Kết quả	5
3.1. Xây dựng chương trình	5
3.1.1. Xử lý đầu vào, tạo mã Huffman từ nguồn tin (Việt Anh)	5
3.1.2. Mã hóa bản tin (Nhật).....	9
3.1.3. Giải bản mã (Quang).....	10
3.1.4. Xử lý đầu vào, ra và giao diện (Chiến).....	12
3.2. Xây dựng trường hợp kiểm thử.....	17
3.3. Kết quả của chương trình qua trường hợp kiểm thử.....	17

1. Giới thiệu đề tài

Mã hóa Huffman là một thuật toán mã hóa mạnh mẽ và phổ biến trong lĩnh vực khoa học máy tính và truyền thông. Được phát triển bởi David A. Huffman vào năm 1952, thuật toán này được sử dụng để nén dữ liệu với mục tiêu giảm dung lượng cần thiết để lưu trữ hoặc truyền tải thông tin. Phương pháp mã hóa Huffman dựa trên nguyên tắc gán mã ngắn hóa cho các ký tự xuất hiện thường xuyên hơn, giảm độ dài của mã bình thường so với mã cố định.

Thuật toán Huffman là một ví dụ điển hình của mã hóa không mất thông tin, nơi chúng ta chủ động giảm kích thước của dữ liệu mà không làm thay đổi nội dung thông tin. Bằng cách sử dụng cây Huffman, mỗi ký tự trong tập hợp dữ liệu được biểu diễn bằng một mã nhị phân duy nhất, với các mã ngắn hóa cho các ký tự xuất hiện phổ biến nhất và các mã dài hơn cho các ký tự xuất hiện ít.

Ứng dụng của mã hóa Huffman không chỉ giới hạn trong lĩnh vực lưu trữ dữ liệu, mà còn mở rộng ra nhiều lĩnh vực khác như truyền thông, nén hình ảnh, âm thanh và video. Đối với những ứng dụng yêu cầu hiệu quả cao và tiết kiệm băng thông, mã hóa Huffman đóng vai trò quan trọng trong việc tối ưu hóa kích thước dữ liệu và tăng tốc quá trình truyền tải. Đồng thời, việc nắm vững thuật toán này giúp lập trình viên hiểu rõ về cơ sở lý thuyết của mã hóa và giải mã, đồng thời có thể ứng dụng nó vào các ứng dụng thực tế.

2. Phân tích đề tài

2.1. Thuật toán mã hóa của Huffman

Cho nguồn tin $X = \{x_1, x_2, \dots, x_q\}$ có phân bố xác suất $P(X) = (p_1, p_2, \dots, p_q)$ được nhập từ bàn phím. Thuật toán tìm bộ mã Huffman với cơ số mã r có các ký hiệu mã có giá trị từ 0 đến $r-1$ gồm các bước (tìm các từ mã ứng với mỗi tin của nguồn):

Bước 1: Tính số nguyên $a = (q-r)/(r-1)$. Nếu a không nguyên thì tính giá trị mới của $a' = \text{Int}(a) + 1$. Tiếp theo tính số tin mới của nguồn $q' = r + a'(r-1)$. Số tin ảo có xác suất bằng 0 là $q' - q$. Nếu a nguyên thì $q' = q$.

Bước 2: Xếp các tin của nguồn q' tin theo thứ tự xác suất giảm dần

Bước 3: Nhóm r tin cuối của nguồn thành 1 tin phụ có xác suất bằng tổng xác suất của các tin được nhóm vào. Đánh dấu mỗi tin được nhóm vào bằng một ký hiệu mã khác nhau.

Bước 4: Lặp lại các bước b và c cho đến khi nguồn còn đúng r tin. Đánh dấu mỗi tin của nguồn này bằng 1 ký hiệu mã khác nhau.

Bước 5: Từ mã ứng với mỗi tin là chuỗi các ký hiệu mã dùng đánh dấu chính tin này và các tin phụ chứa nó (nó được nhóm vào tin phụ này) theo thứ tự ngược với thứ tự nhóm

2.2. Phân tích yêu cầu đề tài

Đề tài bao gồm ba yêu cầu:

1. Xây dựng thuật toán mã hóa tin từ nguồn tin
2. Sử dụng bộ mã Huffman để mã hóa một bản tin
3. Sử dụng bộ mã Huffman để giải mã một bản mã

2.3. Phân công nhiệm vụ

Tên thành viên	Nhiệm vụ
Lê Minh Việt Anh	Tạo mã Huffman từ nguồn tin
Nguyễn Mạnh Chiến	Thiết kế giao diện
Nguyễn Trọng Nhật	Giải mã bản mã
Lê Việt Quang	Mã hóa bản tin

3. Kết quả

Chương trình sử dụng các biến tổng thể:

Tên biến	Kiểu	Ý nghĩa
base	int	Cơ số của mã Huffman
Code	unordered_map<string, string>	Chứa nguồn tin và mã Huffman tương ứng (để mã hóa)
Source	nordered_map<string, double>	Chứa nguồn tin, gồm ký hiệu và xác suất xuất hiện của chúng
reverseCode	unordered_map<string, string>	Chứa mã Huffman và nguồn tin tương ứng (để giải mã)
exist	unordered_map<string, bool>	Ghi nhận sự tồn tại của các tin của nguồn

3.1. Xây dựng chương trình

3.1.1. Xử lý đầu vào, tạo mã Huffman từ nguồn tin (Viết Anh)

Input:

- Các cặp giá trị (s,p) trong đó s là giá trị của tin, p là xác suất xuất hiện tin đó
- Cơ số base của mã Huffman

Output:

- Các cặp giá trị (s, c) trong đó s là giá trị của tin, c là mã Huffman tương ứng

3.1.1.1. Xử lý đầu vào

Trước khi nhập dữ liệu, ta sẽ xóa dữ liệu đã có trước đó (nếu tồn tại):

```
void reset_data()
{
    base = 0;
    unordered_map<string, double>().swap(Source);
    unordered_map<string, string>().swap(Code);
}
```

Ta kiểm soát phần đầu vào đảm bảo các điều sau:

- Kết thúc nhập khi tổng xác suất bằng 1
- Yêu cầu nhập lại một tin của nguồn khi tổng xác suất lớn hơn 1

```

void input_source()
{
    reset_data();
    cout << "Input your information source\n";
    while (sum_probs(Source) <= 0.9999)
    {
        cout << "\tInformation " << Source.size() + 1 << ": ";
        string info;
        cin >> info;
        exist[info] = true;
        cout << "\tProbability " << Source.size() + 1 << ": ";
        double prob;
        cin >> prob;
        check_input(info, prob);
    }
    show_info_source();
}

```

Sau khi nhập, ta xem xét việc sinh mã Huffman từ nguồn hợp lệ.

3.1.1.2. Xây dựng mã Huffman từ nguồn tin

Để thuận tiện, ta chuyển sang ngôn ngữ đồ thị dạng cây. Trong bài toán này là cây **base**-phân đầy đủ (là cây mà mỗi nút hoặc không có nút con nào, hoặc có đủ **base** nút con). Mỗi nút chứa cặp giá trị là tin (kiểu **string**) và xác suất xuất hiện của chúng. Ta xây dựng ngược cây **base**-phân đầy đủ như sau:

Các nút lá là cặp giá trị gồm tin của nguồn và xác suất xuất hiện của nó.

Từ **base** nút có xác suất nhỏ nhất ở độ cao i ($i = 0, 1, 2, \dots$), tạo một nút mới là cha của các nút này, có giá trị là chuỗi ghép từ giá trị của các nút con, xác suất bằng tổng các xác suất của các nút con. Mỗi cạnh nối nút cha và nút con sẽ được gán một giá trị từ 0 đến **base-1** theo thứ tự xác định (từ nút con có xác suất nhỏ nhất đến nút con có xác suất nhỏ thứ **base**).

Đến cuối cùng ta được một nút duy nhất, là gốc của cây. Mã Huffman của mỗi tin (mỗi nút lá) được hình thành bởi một đường đi từ gốc đến nút lá chứa tin đó.

Xây dựng chương trình từ thuật toán:

Đầu tiên, ta tạo thêm các tin ảo cho nguồn để có thể xây dựng được cây **base**-phân đầy đủ:

```

cout << "Base of Huffman code: ";
cin >> base;
int a = (num_r_info-base)/(base-1);
if ((num_r_info-base) % (base-1)) a++;
int num_newsource = base + a*(base-1);
int num_i_info = num_newsource - num_r_info;

```

Trong đó **base** là cơ số của mã Huffman, **num_r_info** là số tin thật, **num_i_info** là số tin ảo, **num_newsource** là số tin của nguồn sau khi đã bổ sung các tin ảo.

Ta sử dụng hàng đợi ưu tiên giá trị nhỏ hơn, để thuận tiện cho việc tìm **base** giá trị nhỏ nhất, và **unordered_map adj** để lưu trữ các nút kề của một nút trong đồ thị. Danh sách kề của một nút sử dụng kiểu dữ liệu **vector**, số phần tử của vector là 1 hoặc 0. Mỗi quan hệ kề của hai nút được định nghĩa: *Nếu nút B là cha của nút A, ta thêm nút B vào danh sách kề của A.*

```

priority_queue <pair <string, double>, vector <pair <string, double>>, CustomComparator> source;
for (auto pair : Source)
{
    source.push({pair.first, pair.second});
}

```

(Bổ sung các tin thật vào hàng đợi ưu tiên)

```

for (int i = 1; i <= num_i_info; i++)
{
    source.push({"i" + to_string(i+1), 0});
}

```

(Bổ sung các tin ảo, là các cặp giá trị ("i0", 0), ..., ("ia", 0) vào hàng đợi ưu tiên)

Tiếp theo, ta xây dựng cây **base**-phân đầy đủ bằng cách thêm các nút mới vào danh sách kề của các nút. Đồng thời, các nút mới này được tạo được bổ sung vào hàng đợi ưu tiên, các nút con của nó sẽ bị xóa khỏi hàng đợi.

```

//tao cay base_phan day du
while (source.size() > 1)
{
    string tmp_info = base_min_label(source);
    double tmp_prob = base_min_sum(source);

    for (int i = 0; i < base; i++)
    {
        adj[source.top().first].push_back({tmp_info, to_string(i)});
        source.pop();
    }
    source.push({tmp_info, tmp_prob});
}

//tinh tong base so nho nhat trong hang doi uu tien
double base_min_sum(priority_queue <pair <string, double>, vector <pair <string, double>>, CustomComparator> q)
{
    priority_queue <pair <string, double>, vector <pair <string, double>>, CustomComparator> qq = q;
    double sum = 0;
    for (int i = 1; i <= base; i++)
    {
        sum += qq.top().second;
        qq.pop();
    }
    return sum;
}

//gan nhan cho base so nho nhat trong hang doi uu tien
string base_min_label(priority_queue <pair <string, double>, vector <pair <string, double>>, CustomComparator> q)
{
    priority_queue <pair <string, double>, vector <pair <string, double>>, CustomComparator> qq = q;
    string label = "";
    for (int i = 1; i <= base; i++)
    {
        label += qq.top().first;
        qq.pop();
    }
    return label;
}

```

(Các hàm **base_min_label**, **base_min_sum** dùng để tạo nhãn và xác suất cho các nút mới)

Cuối cùng, ta xây dựng mã Huffman bằng việc đi từ nút lá về gốc, thông qua danh sách kê:

```

//doc tu goc xuong la de tim ma huffman cho la
for (auto pair : Source)
{
    string p = pair.first;
    while (adj[p].size())
    {
        Code[pair.first] = adj[p].front().second + Code[pair.first];
        p = adj[p].front().first;
    }
}

```

Tạo bảng **reverseCode** để sử dụng cho việc giải bản mã:


```

for (auto pair : Code)
{
    reverseCode[pair.second]=pair.first;
    cout << " " << pair.first << "\t\t" << pair.second << '\n';
    cout << "+-----+\n";
}

```

3.1.2. Mã hóa bản tin (Nhật)

Input:

- Bản tin sequence chứa các kí tự nằm trong bộ mã huffman đã được mã hóa

Output:

- Bản mã huffman_sequence

Sử dụng biến string sequence – để chứa bản tin cần mã hóa; và biến string huffman_sequence để chứa bản tin sau khi mã hóa.

Tạo thủ tục input_sequence() nhập bản tin đầu vào từ bàn phím.

```

void input_sequence(){
    cout << "Input sequence is: ";
    getline(cin,sequence);
}

```

Tuy nhiên, khi nhập bản tin đầu vào thì có thể nhập sai (trong bản tin có chứa kí tự không xuất hiện trong nguồn tin đã cho, nên phải tạo hàm để kiểm tra điều này). Trong quá trình nhập nguồn tin thì ghi nhận sự tồn tại của các kí tự bằng unordered_map <string,bool> exist với trường [key] là kí tự đó và trường [value] là xác nhận sự tồn tại của nó (true: đã tồn tại, false: chưa tồn tại)

```

bool check(string sequence)
{
    for (int i =0; i<sequence.length(); i++)
    {
        if (exist[sequence.substr(i,1)] == false)
            return false;
    }
    return true;
}

```

Hàm check sẽ trả về true nếu trong bản tin nhập vào không chứa kí tự chưa được khai báo trong nguồn tin.

Thủ tục xử lý và tạo mã Huffman:

```
void sequence_process(){
    input_sequence();
    while (!check(sequence)){
        cout<<"Your sequence contains (an) unidentified character(s). Please check again!\n";
        input_sequence();
    }

    huffman_sequence="";
    for (int i=0;i< sequence.length(); i++){
        string tmp = sequence.substr(i,1);
        huffman_sequence += Code[tmp];
    }
    cout<<"Ma huffman la:" << huffman_sequence<<endl;
}
```

Người dùng sẽ phải nhập đúng bản tin đầu vào, nếu không phải nhập lại. Sau khi nhập được bản tin đầu vào, chương trình sẽ lấy ra từng kí tự trong bản tin đó, ánh xạ sang từ mã, rồi nối vào chuỗi huffman_sequence chứa các từ mã tạo thành bản mã

3.1.3. Giải bản mã (Quang)

```
string mahuffman; //Luu tru ma huffman
void input() //ham nhap ma huffman
{
    cout<< "\nEnter Huffman Code: ";

    cin.ignore();
    cin>> mahuffman;
}
//giai ma huffman
int decodeMessage(string code)
```

```
//giai ma huffman
int decodeMessage(string code)
{
    string decodedMessage = "";
    int d=code.length();
    string str= "";
    for(int i=0; i<d; ++i)
    {
        str=str+code[i];
        auto it= reverseCode.find(str);

        if(it!=reverseCode.end())
        {
            decodedMessage=decodedMessage+it->second;
            str="";
        }
    }

    if(str=="") {
        cout<< "Huffman decode: "<<decodedMessage<<endl<<"\n";
        return 1;
    }
    else cout<<"Nhap lai ma huffman:\n";
    return 0;
}
```

- Hàm giải mã với tham số đầu vào là mã huffman
- Hàm khởi tạo 1 chuỗi trống “str” sau đó duyệt qua từng ký tự trong chuỗi mã huffman và thêm ký tự đó vào chuỗi trống ấy
- Duyệt đến khi nào tìm thấy mã huffman tương ứng với mã “str” thì ta lưu ký tự tương ứng với mã huffman ấy vào 1 chuỗi “ string decodedMessage”, rồi trả chuỗi “str” về chuỗi trống.
- Sau đó lại tiếp tục duyệt tiếp đến khi kết thúc rồi in ra chuỗi sau khi giải mã thành công.
- Nếu duyệt đến cuối mà str vẫn còn giá trị trong đó thì yêu cầu “Nhap lai ma huffman” sẽ được hiển thị.

3.1.4. Xử lý đầu vào, ra và giao diện (Chiến)

```

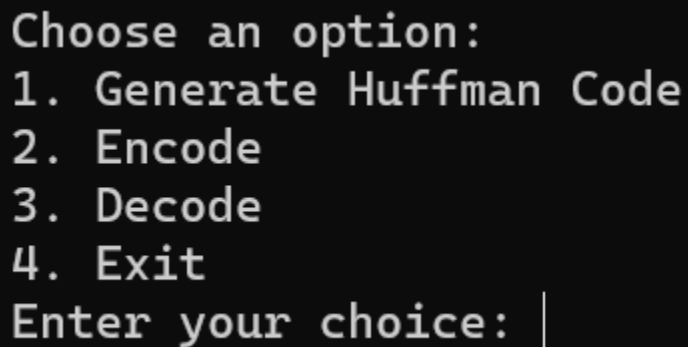
void displaymenu()
{
    cout << "Choose an option:\n";
    cout << "1. Generate Huffman Code\n";
    cout << "2. Encode\n";
    cout << "3. Decode\n";
    cout << "4. Exit\n";
}

void inputmenu()
{
    cout << "\nChoose an option\n";
    cout << "1. Input from keyboard\n";
    cout << "2. Input from file\n";
    cout << "3. Back\n";
}

```

- Menu:
 - 1. Tạo bảng mã Huffman
 - 2. Mã hoá
 - 3. Giải mã
 - 4. Thoát chương trình
- Menu tạo bảng mã có 2 kiểu nhập vào dữ liệu
 - 1. Dữ liệu nhập vào từ bàn phím
 - 2. Dữ liệu nhập vào từ 1 file
 - 3. Trở về Menu ban đầu

```
while(cin)
{
    displaymenu();
    cout << "Enter your choice: ";
    cin >> option;
    switch (option)
    {
        case 1:
            while(cin)
            {
                break;
            }
        case 2:
            sequence_process();
            break;
        case 3:
            input();
            decodeMessage(mahuffman);
            break;
        case 4:
            cout << "Exiting...\n";
            exit(0);
        default:
            cout << "\nInvalid option. Try again.\n";
    }
}
```



```
Choose an option:
1. Generate Huffman Code
2. Encode
3. Decode
4. Exit
Enter your choice: |
```

- Hàm sẽ lặp đi lặp lại cho đến khi người dùng lựa chọn 4 thoát chương trình

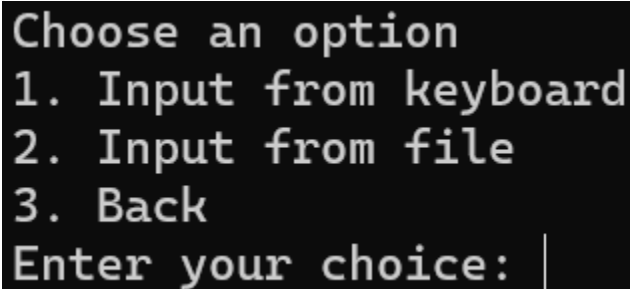
3.1.4.1. Tạo mã Huffman

- Đối với lựa chọn đầu tiên, người dùng sẽ được đưa tới menu lựa chọn phương thức nhập

```

case 1:
    while(cin)
    {
        inputmenu();
        Back=false;
        cout << "Enter your choice: ";
        cin>>choice;
        switch(choice)
        {
            case 1:
                input_processor();
                cout << "\nGenerating Huffman Code...\n";
                show_huffman_code();
                break;
            case 2:
                cout<<"Enter file name: ";
                cin>>filename;
                read_source_from_file(filename);
                cout << "\nGenerating Huffman Code...\n";
                generate_huffman_code();
                show_huffman_code();
                break;
            case 3:
                Back=true;
                break;
            default:
                cout << "\nInvalid option. Try again.\n";
        }
        if(Back)
            break;
    }
    break;

```



```

Choose an option
1. Input from keyboard
2. Input from file
3. Back
Enter your choice: |

```

- Tương tự như trên, menu này sẽ lặp lại cho đến khi người dùng chọn lựa chọn 3 là trở về menu ban đầu.
- Nếu người dùng lựa chọn phương thức nhập từ bàn phím, thì sẽ nhập vào information sau đó đến probability cho đến khi tổng probability bằng 1, sau đó sẽ nhập vào base. Khi thực hiện xong chương trình sẽ bắt đầu tạo mã Huffman từ dữ liệu nhập vào.
- Ngoài ra chương trình còn tính toán thời gian thực hiện và in ra cùng với bảng mã.

PROJECT I: CÁC THUẬT TOÁN NÉN NGUỒN RỜI RẠC

```
Input your information source
Information 1: A
Probability 1: 0.2
Information 2: B
Probability 2: 0.2
Information 3: C
Probability 3: 0.2
Information 4: D
Probability 4: 0.2
Information 5: E
Probability 5: 0.2
Total probability is 1. You have finished inputing. Your source is below:
+-----+
| Information | Probability |
+-----+
| E           | 0.2        |
+-----+
| D           | 0.2        |
+-----+
| C           | 0.2        |
+-----+
| A           | 0.2        |
+-----+
| B           | 0.2        |
+-----+
Base of Huffman code: |
```

```
void read_source_from_file(const string& filename) {
    ifstream file(filename);

    if (!file.is_open()) {
        cerr << "Error opening file: " << filename << endl;
        return;
    }

    reset_data(); // Clear previous data

    string info;
    double prob;

    while (file >> info >> prob) {
        exist[info] = true;
        check_input(info, prob);
    }

    file.close();
    show_info_source();
}
```

- Người dùng lựa chọn phương thức nhập vào từ file thì sẽ phải nhập vào tên của file và trong file sẽ có định dạng như information 1 probability 1 information 2 probability 2 ...:

```
c
0.2
h
0.2
i
0.2
e
0.2
n
0.2
```

- Khi đó chương trình sẽ tiến hành kiểm tra và thực hiện như phương thức thứ nhất
- Trong trường hợp file không tồn tại thì sẽ in ra lỗi và yêu cầu nhập lại

```
Enter file name: chien.txt
Error opening file: chien.txt
```

3.1.4.2. Mã hoá

```
Enter your choice: 2

Input sequence is: chien
```

- Người dùng sẽ được yêu cầu nhập vào chuỗi ký tự cần mã hoá
- Nếu trong đó chứa ký tự chưa được mã hoá thì sẽ thông báo lỗi

```
Your sequence contains (an) unidentified character(s). Please check again!
```

- Nếu không thì sẽ trả về mã hoá của dữ liệu nhập vào

3.1.4.3. Giải mã

```
Enter Huffman Code: 434324
Nhập lại mã huffman:

Enter Huffman Code: 110111100
Huffman decode: abcd
```

- Người dùng sẽ được yêu cầu nhập vào một đoạn mã hoá
- Chương trình sẽ thực hiện giải mã nó và trả về chuỗi ký tự đã được giải mã.
- Nếu mã nhận vào không hợp lệ thì chương trình sẽ yêu cầu nhập lại.

3.2. Xây dựng trường hợp kiểm thử

Input	Expected Output	Result
1	Hiển thị menu nhập nguồn tin	OK
1 a 0.5 b 0.3 c 0.15 d 0.05	Hiển thị bảng cho nguồn tin và xác suất	OK
2	Hiển thị bảng mã Huffman	OK
3	Quay lại menu chính	OK
2 abcdcba	011101100101110	OK
3 011101100101110	abcdcba	OK
4	Kết thúc chương trình	OK

3.3. Kết quả của chương trình qua trường hợp kiểm thử

Qua một số trường hợp kiểm thử, chương trình cho ra kết quả đúng với mong đợi. Thuật toán được cài đặt thành công. Dưới đây là một số thống kê:

- Số dòng lệnh: 388
- Số hàm được dùng: 19
- Kích thước chương trình: 11 KB

-----HẾT-----