TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

PROJECT 1

ĐỀ TÀI: Ứng dụng thuật toán Burrows-Wheeler Transform để nén dữ liệu

Giáo viên hướng dẫn: ***TS. Trần Vĩnh Đức***

Sinh viên thực hiện: Đỗ Lương Kiên - 20183568

HÀ NỘI – 1/2021

Mục lục

CHƯƠNG 1: Thuật toán Burrows-Wheeler Transform

1.1. Giới thiệu

Các thuật toán nén dữ liệu được sử dụng rộng rãi nhất được dựa trên nén dữ liệu tuần tự của Lempel Ziv. Kỹ thuật nén Burrows Wheeler không xử lý dữ liệu đầu vào theo tuần tự, thay vào đó là xử lý một khối văn bản đầu vào như một đơn vị duy nhất. Ý tưởng của chuyển đổi Burrows Wheeler (Burrows Wheeler Transform - BWT) là hoán vị các chữ cái trong một tài liệu là m cho nhiều ký tự trong văn bản được chuyển đổi xuất hiện trong các *run* (xâu gồm các biểu tượng giống nhau), hoặc rất gần với sự xuất hiện trước đó để dễ dàng hơn cho việc thực hiện nén.

1.2. Chuyển đổi Burrows-Wheeler thuận

Giả sử BWT mã hóa xâu T gồm n kí tự T[1..n]. trên một bảng chữ cái ∑ gồm |∑| kí tự. Chuyển đổi thuận về bản chất liên quan đến việc sắp xếp tất cả các phép quay của xâu đầu vào, nhóm các ký tự xuất hiện trong các ngữ cảnh tương tự lại với nhau.

☞ **Bước 1**: Xâu cần mã hóa được dịch chuyển vòng tròn và tạo thành một ma trận L*L (với L là độ dài xâu kí tự)

☞ **Bước 2**: Sắp xếp lại các dòng của ma trận theo thứ tự từ điển

☞ **Bước 3**: Trích xâu từ các kí từ cuối ở mỗi dòng, thông báo xâu này và cho biết từ gốc là thứ tự thứ mấy trong ma trận nhận được ở bước 2

Ví dụ: Ta cần mã hóa xâu BANANA

Bước 1: Xâu cần mã hóa được dịch chuyển vòng tròn và tạo thành một ma trận nxn (với n là độ dài xâu kí tự).

B	A	N	A	N	A
A	N	A	N	A	B
N	A	N	A	B	A
A	N	A	B	A	N
N	A	B	A	N	A
A	B	A	N	A	N

Hình 1.1a: Mảng A chứa tất cả các phép quay đầu vào của xâu BANANA

Bước 2: Sắp xếp lại các dòng của ma trận theo thứ tự từ điển

A	B	A	N	A	**N**
A	N	A	B	A	**N**
A	N	A	N	A	**B**
B	**A**	**N**	**A**	**N**	**A**
N	A	B	A	N	**A**
N	A	B	A	N	**A**

Hình 1.1b: Ma trận A_s thu được bằng các sắp xếp A.

Cột cuối của A_s là đầu ra của BWT

Bước 3: Trích xâu từ các kí từ cuối ở mỗi dòng, thông báo xâu này và cho biết từ gốc là thứ tự thứ mấy trong ma trận nhận được ở bước 2.

Ta có (NNBAAA,4). Xâu NNBAAA chính là xâu để thực hiện nén trong các bước tiếp theo.

Tuy nhiên thay vì sử dụng cách trên, ta có thể dựa vào Suffix Array – mảng hậu tố để tính BWT của 1 xâu (đây cũng là cách mà được em sử dụng), gồm các bước sau:

- **Bước 1**: Tính mảng hậu tố (Suffix Array ký hiệu SA[]) của xâu đầu vào
- **Bước 2**: Sắp xếp mảng hậu tố theo thứ tự từ điển
- **Bước 3:** Dựa vào mảng hậu tố, tính BWT của xâu theo công thức:

$$BWT[i] = T[(SA[i] + n - 1)\%n] \quad \text{với mọi i từ 0 → n-1}$$

Trong đó
- BWT[]: là mảng chứa xâu sau khi được chuyển đổi
- T[]: xâu ban đầu
- SA[]: mảng hậu tố
- n: độ dài của xâu ban đầu

Ví dụ: Ta biến đổi xâu T = "BANANA"

Bước 1: Các hậu tố:

0	BANANA
1	ANANA
2	NANA
3	ANA
4	NA
5	A

Bước 2: Sắp xếp theo thứ tự từ điển

5	A
3	ANA
1	ANANA
0	BANANA
4	NA
2	NANA

Sau bước 2, ta được mảng SA[] = {5, 3, 1, 0, 4, 2}

Bước 3: Áp dụng công thức $BWT[i] = T[(SA[i] + n - 1)\%n]$ để tính BWT:

BWT = [N, N, B, A, A, A]

Nhận xét: Với cách chuyển đổi trên, thông thường thời gian sắp xếp trung bình là *O(nlogn)* nếu phương pháp *quick sort* được sử dụng.

Như ta đã thấy: Chuyển đổi Burrows Wheeler không làm cho chuỗi BANANA ngắn đi, nhưng lại làm xuất hiện các *run* (xâu gồm các ký tự giống nhau hoặc rất gần với sự xuất hiện trước đó).

Ưu điểm: Sự phân cụm này làm cho quá trình nén trở nên dễ dàng hơn.

1.3. Kỹ thuật nén dữ liệu Burrows-Wheeler

Ở phần 1.2, ta thấy chuyển đổi Burrows-Wheeler đã làm cho một văn bản được chuyển đổi và xuất hiện các run (xâu gồm các ký tự giống nhau hoặc rất gần với sự xuất hiện trước đó).

Chuyển đổi Burrows-Wheeler không làm giảm chiều dài của xâu chuyển đổi mà chỉ làm cho xâu sau khi chuyển đổi dễ thực hiện các phương pháp nén đã có như: mã hóa RLE, mã hóa Huffman, …

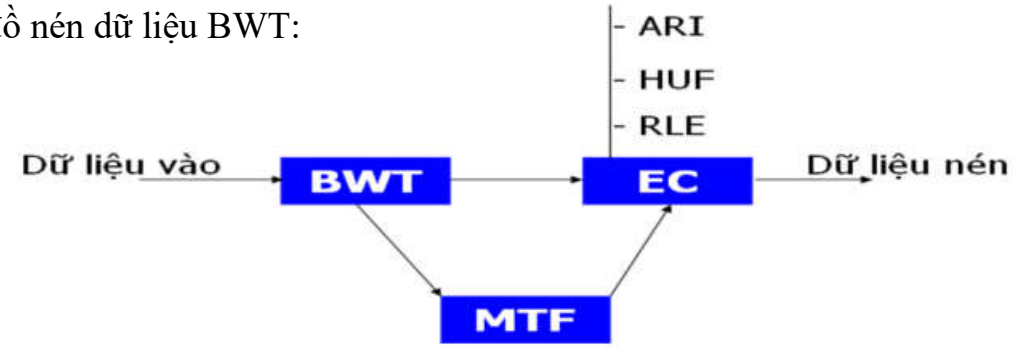
Năm 1994, Burrows và Wheeler giới thiệu chuyển đổi Burrows và Wheeler (BWT). Một thực hiện cũng đã được biết đến của thuật toán này là Bzip2. Chương trình này thường co lại lại một văn bản Tiếng Anh khoảng 20% kích thước ban đầu của nó trong khi gzip chỉ giảm xuống khoảng 26% kích thước ban đầu. Thuật toán bao gồm 3 bước cần thiết sau đây:

1. **Burrows Wheeler Transform (BWT):** Hoán vị các chữ cái trong một tài liệu theo từng khối (khoảng 200 KB) để có một văn bản được chuyển đổi bao gồm các run (xâu gồm các biểu tượng giống nhau) hoặc gần với sự xuất hiện trước đó.

2. **Di chuyển đến phía trước để mã hóa (Move To Front):** Bước này chuyển đổi file văn bản đã được chuyển đổi Burrows Wheeler thành một văn bản có xu hướng làm tăng dần tần số của các biểu tượng giá trị thấp trong một khối để cải thiện việc nén của các bộ mã entropy như: Mã Huffman hoặc mã số học.

3. **Sử dụng mã Huffman** để nén luồng những chỉ số từ giai đoạn MTF thành luồng bit có kích thước nhỏ.

Lược đồ nén dữ liệu BWT:



CHƯƠNG 2: Chuyển đổi lên trước (Move-to-front)

2.1. Giới thiệu

Chuyển đổi lên trước(MTF) là mã hóa dữ liệu (thường là một luồng byte) được thiết kế để cải thiện hiệu suất của các kỹ thuật mã hóa entropy nén. Nó là một bước bổ sung trong thuật toán nén dữ liệu.

2.2. Move-to-front

Ý tưởng chính của thuật toán là mỗi ký tự trong dữ liệu được thay thế bằng chỉ mục của nó trong ngăn xếp của "Chuỗi ký tự được sử dụng". Ví dụ, chuỗi các ký tự giống nhau mà đứng cạnh nhau được thay thế bằng chuỗi ký tự 0, trong khi một ký tự không được sử dụng một thời gian dài xuất hiện, nó được thay thế bằng một số lớn.

Các bước thực hiện:

- Bước 1: Tìm vị trí của ký tự đang được xét trong chuỗi ký tự được sử dụng.
- Bước 2: Lưu vị trí đó
- Bước 3: Chuyển ký tự đang được xét lên đầu chuỗi ký tự được sử dụng
- Bước 4: Lặp lại bước 1 cho tới khi hết các ký tự.

Ví dụ: xét chuỗi đầu vào T = nnbaaa

Giả sử danh sách ký tự sử dụng là List = (abcdefghijklmnopqrstuvwxyz)

Chuỗi đầu vào	Chuỗi ký tự sử dụng	Kết quả
nnbaaa	(abcdefghijklm**n**opqrstuvwxyz)	13
n**n**baaa	(**n**abcdefghijklmopqrstuvwxyz)	13, 0
nn**b**aaa	(na**b**cdefghijklmopqrstuvwxyz)	13, 0, 2
nnb**a**aa	(bn**a**cdefghijklmopqrstuvwxyz)	13, 0, 2, 2
nnba**a**a	(**a**bncdefghijklmopqrstuvwxyz)	13, 0, 2, 2, 0
nnbaa**a**	(**a**bncdefghijklmopqrstuvwxyz)	13, 0, 2, 2, 0, 0
Final	(abncdefghijklmopqrstuvwxyz)	13, 0, 2, 2, 0, 0

Bằng cách này khi kết thúc dãy, đầu ra cuối cùng ta thu được kết quả:

MTF = [13, 0, 2, 2, 0, 0]

Với mã hóa MTF, Các ký tự thứ hai và liên tiếp trong bất kỳ *run* nào được chuyển đổi thành 0 (“NN” đã trở thành 13,0; “AAA” trở thành 2,0,0). Điều này hoạt động tốt với BWT, vì nó sinh ra các khối với nhiều *run.*

Ưu điểm: Giai đoạn MTF quản lý một danh sách gồm 95 ký tự (có thể in ra màn hình), và xử lý tuần tự các ký tự đầu vào. Điều này làm cho những ký tự thường xảy ra trong thời gian gần được chuyển thành các chỉ số nhỏ. Những *run* của những ký tự như nhau được chuyển thành *run* của các số 0, điều này làm xuất hiện nhiều giá trị 0 trong dãy đầu ra và làm cho MTF hữu ích để cải thiện việc nén.

Trong bước này, em cũng đã thực hiện việc tính tần số xuất hiện của các ký tự để làm cơ sở cho bước 3: Mã hóa huffman

CHƯƠNG 3: Mã hóa Huffman

3.1. Giới thiệu

Mã hóa huffman là một thuật toán mã hóa dùng để nén dữ liệu. Nó dựa trên bảng tần suất xuất hiện các ký tự cần mã hóa để xây dựng một bộ mã nhị phân cho các ký tự đó sao cho dung lượng (số bit) sau khi mã hóa là nhỏ nhất.

3.2. Giải thuật tham lam xây dựng cây Huffman

Trong giải thuật tham lam để xây dựng cây mã tiền tố tối ưu của Huffman, ở mỗi bước ta chọn 2 ký tự có tần số thấp nhất để mã hóa bằng từ mã dài nhất.

Giả sử có tập A gồm *n* ký hiệu và hàm trọng số tương *ứng W(i), i = 1..n.*

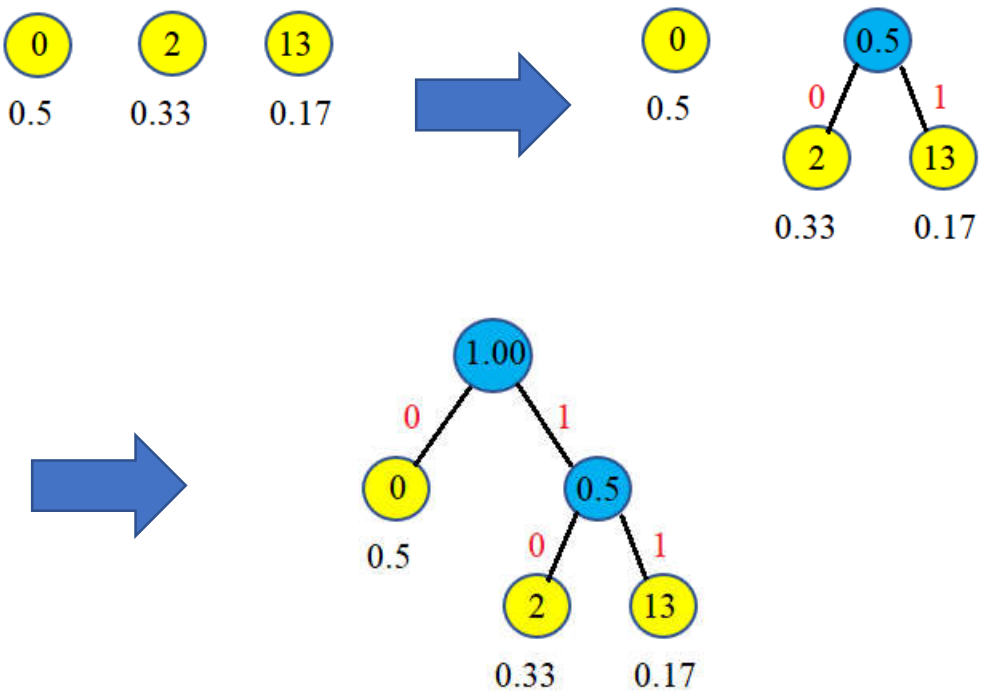
- Khởi tạo: Tạo một rừng gồm n cây, mỗi cây chỉ có một nút gốc, mỗi nút gốc tương ứng với một ký tự và có trọng số là tần số/tần suất của ký tự đó *W(i).*
- Lặp:
    - Mỗi bước sau thực hiện cho đến khi rừng chỉ còn một cây:
    - Chọn hai cây có trọng số ở gốc nhỏ nhất hợp thành một cây bằng cách thêm một gốc mới nối với hai gốc đã chọn. Trọng số của gốc mới bằng tổng trọng số của hai gốc tạo thành nó.

Như vậy ở mỗi bước số cây bớt đi một. Khi rừng chỉ còn một cây thì cây đó biểu diễn mã tiền tố tối ưu với các ký tự đặt ở các lá tương ứng.

Ví dụ: Cho 1 chuỗi nhãn là: 13,0,2,2,0,0

Cho bảng tần xuất của 3 ký tự ‘0’, ‘2’, ‘13’ tương ứng là 0.5; 0.33; 0.17

Quá trình xây dựng cây huffman diễn ra như sau:



Như vậy, bộ mã tối ưu tương ứng là: '0': 0; '2': 10; '13': 11.

3.3. Nén file bằng mã huffman

Trong các bước trên, giả sử đã xây dựng được bộ mã Huffman của 256 ký hiệu có mã ASCII từ 0 đến 255 chứa trong mảng Code[1..256]. Việc nén file có thể phân tích sơ bộ như sau:

1. Đọc từng byte của file cần nén cho đến khi hết tệp,
2. Chuyển theo bộ mã thành xâu nhị phân,
3. Ghép với xâu nhị phân còn dư từ bước trước,
4. Nếu có đủ 8-bit trong xâu thì cắt ra 8-bit đầu tiên ghi vào tệp nén,
5. Nếu đã hết tệp cần nén thì dừng...

CHƯƠNG 4: Kết quả thực hiện

4.1. Cài đặt chương trình

Trong chương trình này, em đã cài đặt trên ngôn ngữ C để xây dựng chương trình thử nghiệm áp dụng thuật toán nén huffman kết hợp với BWT và MTF.

Ý tưởng của chương trình như sau:

i. Đọc chuỗi ký tự đầu vào từ file (1 chuỗi có kích thước ~ 200Kb)
ii. Với mỗi chuỗi đó, thực hiện BWT, MTF và tính luôn tần suất của các ký tự sau khi được biến đổi trong quá trình MTF. Ghi kết quả sau khi MTF ra file tmp.txt
iii. Thực hiện bước 2 cho tới khi đọc hết file.
iv. Xây dựng cây Huffman và nén dữ liệu.

4.2. Kết quả thu được

Link Github mã nguồn: kien1752000/project_1: using Burrows-Wheeler Transform to compression data (github.com)

Sau khi chạy thử với 1 số file đầu vào mà em tự chuẩn bị, chương em thu được kết quả như sau:

- Với file 1MB, thời gian chạy chương trình ~ 1s.
- Với file 10MB, thời gian chạy chương trình ~ 10s.
- Với file 100MB, thời gian chạy chương trình ~ 95s.
- Với file 500MB, thời gian chạy chương trình ~ 472s.

4.3. Nhận xét

Qua kết quả thu được ở trên, em nhận thấy rằng chương trình của mình chưa đủ tốt để có thể áp dụng vào thực tế. Cần phải cải tiến nhiều để có thể thực hiện điều này. Tuy nhiên, em cũng đã cố gắng rất nhiều để hoàn thiện và thu được kết quả thử nghiệm như trên.

TÀI LIỆU THAM KHẢO

1. Burrows–Wheeler transform - Wikipedia
2. Move-to-front transform - Wikipedia
3. Huffman coding - Wikipedia
4. Computing the Burrows–Wheeler transform in place and in small space - ScienceDirect
5. COS 226 Burrows-Wheeler Data Compression Algorithm (princeton.edu)
6. Data struct and Algorithms: Bàn về mảng hậu tố (Suffix Array) (blogthuattoan.blogspot.com)
7. Huffman Coding | Greedy Algo-3 - GeeksforGeeks