ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN CUỐI KỲ

MÔN HỌC: TÍNH TOÁN ĐA PHƯƠNG TIỆN - CS232.N21

ĐỀ TÀI

TEXT COMPRESSION USE ARITHMETIC CODING AND ADAPTIVE HUFFMAN

GV hướng dẫn: THS. ĐỖ VĂN TIẾN

Nhóm thực hiện:

1. Trần Văn Lực - 20521587
2. Ngô Ngọc Sương - 20521852
3. Nguyễn Văn Hợp - 20521358

Thành phố Hồ Chí Minh năm 2023

MỤC LỤC

MỤC LỤC

Chương 1. TỔNG QUAN

1.1. Giới thiệu bài toán

Trong thế giới hiện đại, việc truyền tải và lưu trữ dữ liệu là một yếu tố quan trọng trong nhiều ứng dụng. Với sự gia tăng đáng kể về lượng thông tin được tạo ra hàng ngày, việc nén dữ liệu trở thành một thách thức quan trọng. Trong lĩnh vực tính toán đa phương tiện, nén văn bản là một trong những bài toán quan trọng và được nghiên cứu rộng rãi. Bài toán nén văn bản nhằm giảm kích thước của văn bản ban đầu mà vẫn đảm bảo khôi phục lại nội dung gốc một cách chính xác. Mục tiêu chính của việc nén văn bản là tiết kiệm không gian lưu trữ và tối ưu hóa quá trình truyền tải thông tin qua mạng. Trong đồ án này, nhóm chúng tôi tập trung vào hai phương pháp nén văn bản phổ biến là mã hóa dựa trên Arithmetic Coding và Adaptive Huffman.

Phương pháp Arithmetic Coding dựa trên việc mã hóa văn bản thành một chuỗi số thực trong khoảng từ 0 đến 1, sử dụng xác suất xuất hiện của các ký tự. Phương pháp này có khả năng nén văn bản với mức độ nén tốt và khả năng khôi phục nội dung gốc một cách chính xác. Phương pháp Adaptive Huffman là một phương pháp mã hóa dựa trên cây Huffman, trong đó các ký tự xuất hiện phổ biến được mã hóa bằng các mã ngắn hơn và các ký tự xuất hiện ít được mã hóa bằng các mã dài hơn. Điều đặc biệt về phương pháp này là nó thích nghi với thông tin đầu vào, cho phép xây dựng cây Huffman linh hoạt và thay đổi khi có sự thay đổi trong tần số xuất hiện của các ký tự. Tuy nhiên, việc nén văn bản không chỉ đòi hỏi hiệu quả mà còn đặt ra một số thách thức. Đối với các ứng dụng thời gian thực hoặc có băng thông hạn chế, tốc độ nén và giải nén là yếu tố quan trọng. Do đó, trong quá trình nghiên cứu và thực hiện, chúng tôi sẽ tập trung vào cả khả năng nén hiệu quả và hiệu suất của hệ thống.

Qua đồ án này, nhóm hy vọng rằng chúng tôi sẽ có được cái nhìn sâu hơn về hai phương pháp nén văn bản - Arithmetic Coding và Adaptive Huffman. Đồng thời,

chúng tôi cũng mong muốn xây dựng một hệ thống nén văn bản hiệu quả và đánh giá hiệu suất của hệ thống thông qua các thử nghiệm trên các bộ dữ liệu văn bản khác nhau.

1.2. Thách thức bài toán

Nén văn bản là một bài toán khó, đặc biệt là khi chúng ta muốn giữ lại nội dung gốc của văn bản trong quá trình nén và giải nén. Thách thức chính của bài toán là tìm cách biểu diễn một văn bản bằng một chuỗi bit có kích thước nhỏ hơn, nhưng vẫn có thể khôi phục lại văn bản ban đầu một cách chính xác. Đồng thời, việc áp dụng các phương pháp nén văn bản cần phải đảm bảo tốc độ nén và giải nén hiệu quả, vì đối với các ứng dụng thời gian thực hoặc có băng thông hạn chế, việc xử lý dữ liệu nhanh chóng là rất quan trọng.

1.3. Mục tiêu và phạm vi

Mục tiêu là nghiên cứu và thực hiện hai phương pháp nén văn bản - Arithmetic Coding và Adaptive Huffman, muốn hiểu rõ cách hoạt động của hai phương pháp này và xác định ưu điểm và hạn chế của từng phương pháp và xây dựng một hệ chương trình nén văn bản sử dụng các phương pháp này, đồng thời đánh giá hiệu suất và chất lượng nén của 2 phương pháp.

Phạm vi bài toán: Bài toán của chúng tôi tập trung vào việc nén văn bản sử dụng hai phương pháp nén - Arithmetic Coding và Adaptive Huffman. Chúng tôi sẽ tạo ra một chương trình để thực hiện quá trình nén và giải nén văn bản. Phạm vi của bài toán không bao gồm các loại dữ liệu khác ngoài văn bản. Chúng tôi sẽ thực hiện các thử nghiệm và so sánh hiệu suất của hai phương pháp nén này dựa trên các bộ dữ liệu văn bản khác nhau.

Chương 2. NỘI DUNG CHÍNH

2.1. Thuật toán Arithmetic Coding

2.1.1. Giới thiệu

Thuật toán Arithmetic được đề xuất lần đầu tiên trong một bài báo năm 1987 (Witten, Ian H., Radford M. Neal, and John G. Cleary. "Arithmetic coding for data compression.").Thuật toán này là một dạng mã hóa entropy được sử dụng trong nén dữ liệu không mất thông tin, cho phép khôi phục lại toàn bộ dữ liệu ban đầu qua các chu trình nén và giải nén. Thuật toán hoạt động bằng cách mã hóa dữ liệu thành một số thực nằm trong một khoảng giá trị cụ thể, sau đó sử dụng số thức đó để giải mã và khôi phục lại dữ liệu ban đầu.

2.1.2. Nguyên lý

Thuật toán biểu diễu dữ iệu dựa trên các khoảng xác suất xuất hiện của các ký tự trong dữ liệu. Ý tưởng chính của thuật toán là tạo ra một khoảng giá trị ban đầu từ 0 đến 1 và tiến hành mã hóa từng ký tự trong dữ liệu bằng cách cập nhật khoảng giá trị theo khoảng xác xuất xuất hiện của các ký tự tương ứng.

2.1.3. Thuật toán

2.1.3.1. Thuật toán nén

- ***Input và Output***
    - Input:
        - Chuỗi cần nén.
        - Một dictionary với key là ký tự và value là khoảng xác xuất tương ứng với ký tự đó.
    - Output: Giá trị đã nén (số thực)
- ***Các bước thực hiện thuật toán Arithmetic Coding***

1. Xác định khoảng con ban đầu (hiện tại): Từ 0.0 đến 1.0
2. Xác định khoảng con cho từng ký tự theo công thức sau:

   $$\mathbf{S + P * R}$$

   Trong đó:

   - S: Tổng tích lũy của xác xuất trước đó
   - P: Khoảng xác xuất của ký tự đó (range_start, range_end)
   - R: khoảng giá trị của khoảng con
3. Cập nhật khoảng con hiện tại
4. Nếu chưa phải ký tự cuối cùng, quay lại bước 2

   Ngược lại, xác định giá trị đã nén. (Giá trị này là giá trị bất kỳ nằm trong khoảng con hiện tại của ký tự cuối cùng)

- ***Để dễ tiếp cận các bước của thuật toán nén, chúng tôi có ví dụ cho chuỗi "abca"***

  Bảng tần số

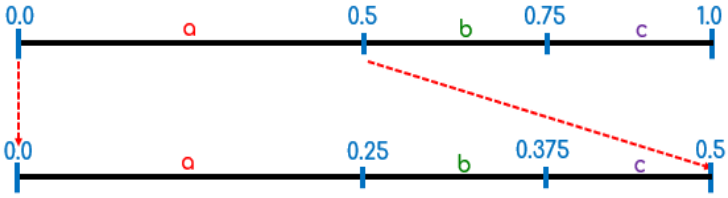
Ký tự	a	b	c
Tần số	2	1	1

  Bảng xác suất

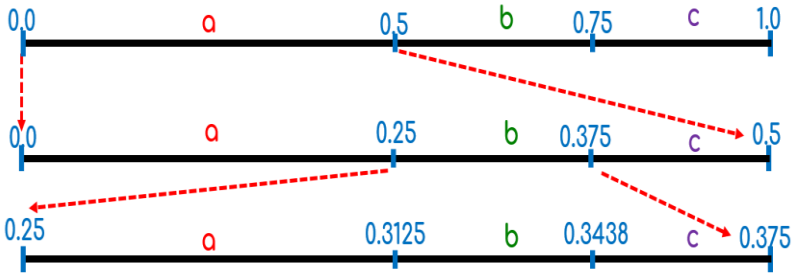
Ký tự	a	b	c
Xác xuất	0.5	0.25	0.25

1. Khoảng con ban đầu: Từ 0 đến 1
2. Cập nhật khoảng con hiện tại: Vì ký tự đầu tiên là a nên khoảng con hiện tại là từ 0.0 đến 0.5.
3. Xác định khoảng con cho từng ký tự, R = 0.5 – 0.0 = 0.5
   - Với ký tự a, nó bắt đầu từ 0.0 đến 0.0 + 0.5*0.5 = 0.25. Khoảng con là (0.0:0.25)

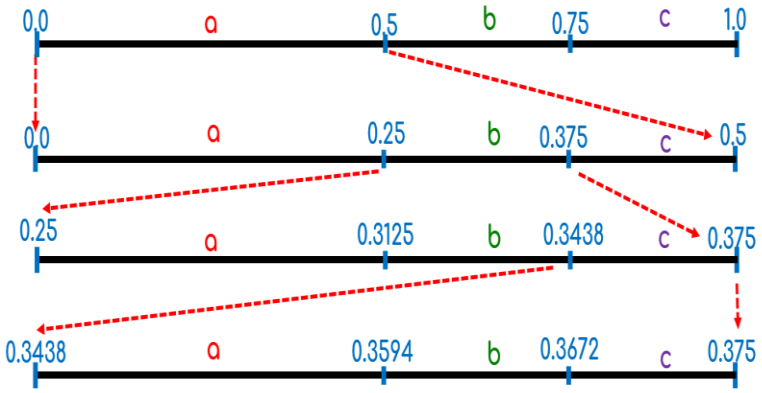
- Với ký tự b, nó bắt đầu từ 0.25 đến 0.25 + 0.25*0.5 = 0.375. Khoảng con là (0.25:0.375)
- Với ký tự c, nó bắt đầu từ 0.375 đến 0.375 + 0.25*0.5 = 0.5. Khoảng con là (0.375:0.5)



4. Cập nhật khoảng con hiện tại: Ký tự tiếp theo là b nên khoảng con hiện tại là từ 0.25 đến 0.375.
5. Quay lại bước 2, R = 0.125. Tương tự:
   - Với ký tự a, khoảng con là (0.25:0.3125)
   - Với ký tự b, khoảng con là (0.3125:0.3438)
   - Với ký tự c, khoảng con là (0.3438:0.375)



6. Cập nhật khoảng con hiện tại: Ký tự tiếp theo là c nên khoảng con hiện tại là từ 0.3438 đến 0.375.
7. Quay lại bước 2, R =0.0312. Tương tự:
   - Với ký tự a, khoảng con là (0.3438:0.3594)
   - Với ký tự b, khoảng con là (0.3594:0.3672)
   - Với ký tự c, khoảng con là (0.3672:0.375)



8. Cập nhật khoảng con hiện tại: Ký tự cuối cùng là a nên khoảng con hiện tại là từ 0.3438 đến 0.3594.
9. Chọn một giá trị đã nén bất kỳ nằm trong khoảng từ 0.3438 đên 0.3594. (Chọn giá trị trung bình => giá trị mã hóa = 0.3516)

2.1.3.2. Thuật toán giải nén

- ***Ý tưởng chính:***

  Duyệt qua từng ký tự bất kỳ và so sánh giá trị nén với khoảng xác xuất để xác định ký tự tương ứng.

- ***Input và Output***
  - Input:
  - Giá trị đã nén
  - Một dictionary với key là ký tự và value là khoảng xác xuất tương ứng với ký tự đó.
  - Số lượng ký tự của chuỗi
  - Output: Chuỗi ban đầu trước khi nén
- ***Các bước thực hiện thuật toán Arithmetic Coding***

  1. Tạo một chuỗi rỗng để lưu trữ dữ liệu giải nén.
  2. Duyệt qua từng ký tự và khoảng xác xuất tương ứng

- Xác định range_width của khoảng xác xuất.
- Kiểm tra giá trị nén có nằm trong khoảng xác xuất của ký tự hiện tại hay không.
    - Nếu có thì thêm vào chuỗi giải nén và cập nhật giá trị nén (giá trị nén –range_start)/range_width.

3. Thực hiện cho đến khi giải nén đủ số ký tự ban đầu.

- ***Để dễ tiếp cận các bước của thuật toán giải nén, chúng em có ví dụ giải nén cho chuỗi "abca" đã nén ở trên***

  Giá trị đã nén = 0.3516, số lượng ký tự = 4

  Bảng xác suất

Ký tự	a	b	c
Xác xuất	0.5	0.25	0.25

Giá trị nén	Output	Low	High	Range
0.3516	a	0.0	0.5	0.5
0.7032	b	0.5	0.75	0.25
0.8128	c	0.75	1.0	0.25
0.2512	a	0.0	0.5	0.5

1. Với ký tự đầu tiên, giá trị nén là 0.3516, nằm trong khoảng (0.0, 0.5) => Xuất ký tự "a"
2. Với ký tự thứ 2, giá trị nén là (0.3516 – 0.0)/0.5 = 0.7032, nằm trong khoảng (0.5, 0.75) => Xuất ký tự "b"
3. Với ký tự thứ 3, giá trị nén là (0.7032 – 0.5)/0.25 = 0.8128, nằm trong khoảng (0.75, 1.0) => Xuất ký tự "c"
4. Với ký tự thứ 4, giá trị nén là (0.8128 – 0.75)/0.25 = 0.2512, nằm trong khoảng (0.0, 0.5) => Xuất ký tự "a"

5. Vì số ký tự trong chuỗi giải nén đã bằng 4 và bằng số ký tự input nên ta dừng và xuất chuỗi đã giải nén “abca”.

2.1.4. Ưu điểm và hạn chế của Arithmetic

- **Ưu điểm**
  - Là thuật toán nén không mất mát thông tin.
  - Hiệu suất nén cao: Arithmetic Coding có khả năng nén dữ liệu với hiệu suất cao hơn so với nhiều thuật toán nén khác. Nó có thể tạo ra kích thước file đầu ra gần với giới hạn lý thuyết của dữ liệu nén.
  - Tính linh hoạt: có thể nén nhiều loại dữ liệu khác nhau như văn bản, hình ảnh, video,…
- **Hạn chế**
  - Tính toán phức tạp: Tính toán với số thực yêu cầu sự chính xác cao và có thể gây tốn thời gian.
  - Nhạy cảm với lỗi: Vì tính toán với số thực, nên một số sai sót trong quá trình nén có thể dẫn đến kết quả giải nén không chính xác
  - Chỉ thực hiện được khi biết tần suất xuất hiện của các ký tự

2.2. Thuật toán Adaptive Huffman Coding

2.2.1. Giới thiệu

Thuật toán Adaptive Huffman Coding là một phương pháp nén dữ liệu hiệu quả, được sử dụng để giảm kích thước của thông tin trước khi được truyền hoặc lưu trữ. Nó là một phiên bản cải tiến của Huffman Coding, mà cho phép xây dựng cây Huffman cơ bản dựa trên thông tin xuất hiện trong quá trình nén. Điều này cho phép thuật toán thích ứng và linh hoạt hơn trong việc xử lý các tần số xuất hiện của các ký tự trong dữ liệu đầu vào.

2.2.2. Nguyên lý

Nguyên lý của thuật toán Adaptive Huffman Coding là xây dựng cây Huffman động dựa trên thông tin xuất hiện của các ký tự trong dữ liệu đầu vào. Ban đầu, cây Huffman chỉ chứa một nút duy nhất chứa ký tự đặc biệt (NYT - Not Yet Transferred). Khi các ký tự khác xuất hiện, chúng được thêm vào cây theo các quy tắc nhất định để đảm bảo cây vẫn cập nhật và phản ánh chính xác tần số xuất hiện của các ký tự. Điều này cho phép thuật toán thích ứng với các thay đổi trong dữ liệu đầu vào và tạo ra mã nén ngắn hơn cho các ký tự xuất hiện thường xuyên.

2.2.3. Thuật toán

2.2.3.1. Thuật toán nén

- ***Input và Output***
    - Input: Chuỗi các ký tự.
    - Output: Chuỗi ký tự ban đầu được biểu diễn bằng các mã nhị phân.

Thuật toán nén Adaptive Huffman Coding bao gồm các bước sau:

Bước 1: Khởi tạo cây Adaptive Huffman với một nút duy nhất chứa ký tự đặc biệt NYT có trọng số bằng 0.

Bước 2: Đọc lần lượt từng ký tự trong dữ liệu đầu vào → mã hóa lần lượt từng ký tự.

- Kiểm tra ký tự có trong cây Adaptive Huffman hay chưa?
    - Nếu có: Phát sinh mã bit cho ký tự (giống như cây Huffman thông thường).
    - Nếu chưa có: Phát sinh mã bit bao gồm "mã bit của nút NYT + mã *ASCII 8-bit* của ký tự". Lưu ý: mã bit gán cho từng ký tự có thể được gán với một mã đã được thỏa thuận ban đầu mà không cần biết trước tần suất xuất hiện của chúng. Tuy nhiên, thông thường mã ASCII được sử dụng để mã hóa các ký tự.

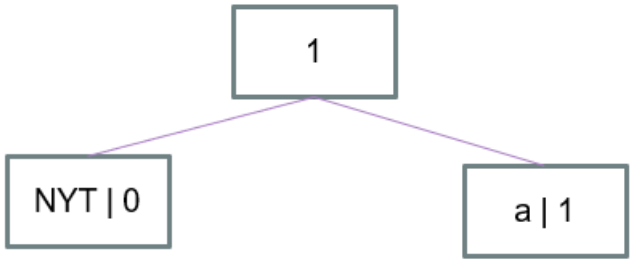
Bước 3: Cập nhật cây Adaptive Huffman.

- Gọi p là nút lá chứa ký tự.
- Nếu ký tự chưa có trong cây → tách nút NYT thành hai nút
  - Nút NYT (mới) có trọng số 0.
  - Và nút lá p chứa ký tự.
- Nếu ký tự đã có trong cây
  - Nếu p là nút gốc thì tăng trọng số lên 1.
  - Hoán đổi p với nút xa nhất (tính từ nút p theo hướng trái sang phải và dưới lên trên) có cùng trọng số (không tính nút cha của p) → tăng trọng số của p lên 1.
- Cập nhật trọng số của các nút liên quan trong cây.
- Kiểm tra lại tính chất anh/em và điều chỉnh lại cây.
  - Các nút không phải nút gốc phải có nút anh em (vì là cây nhị phân đầy đủ).
  - Trọng số tăng dần từ dưới → trên, và từ trái → phải.

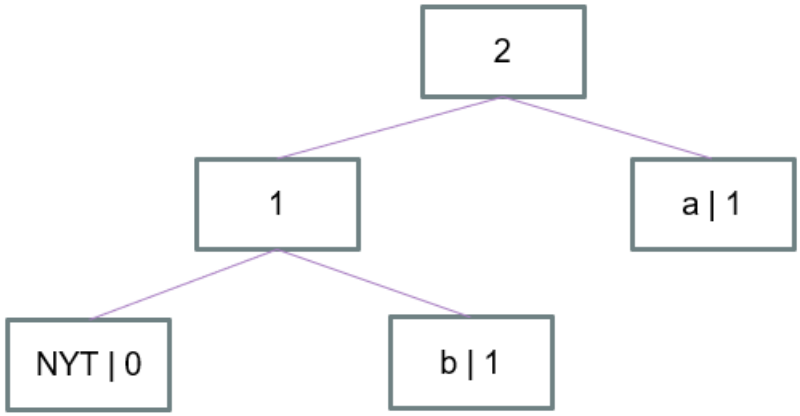
Để dễ tiếp cận các bước của thuật toán nén, chúng em có ví dụ đơn giản như sau: cho chuỗi "abbccd" và chạy từng bước thuật toán.

1. **Khởi tạo cây Adaptive Huffman ban đầu**

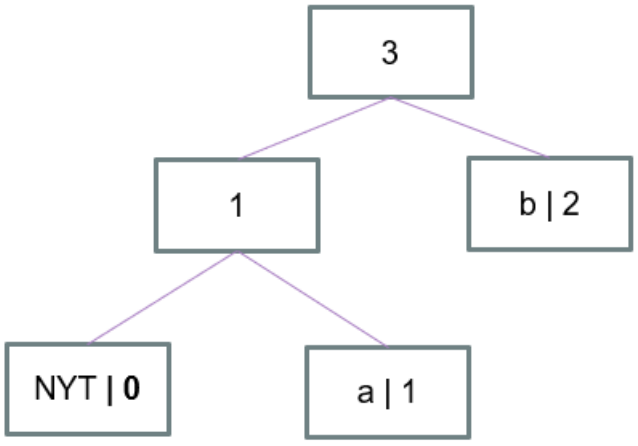
2. **Input = a, output(a) = 01100001**



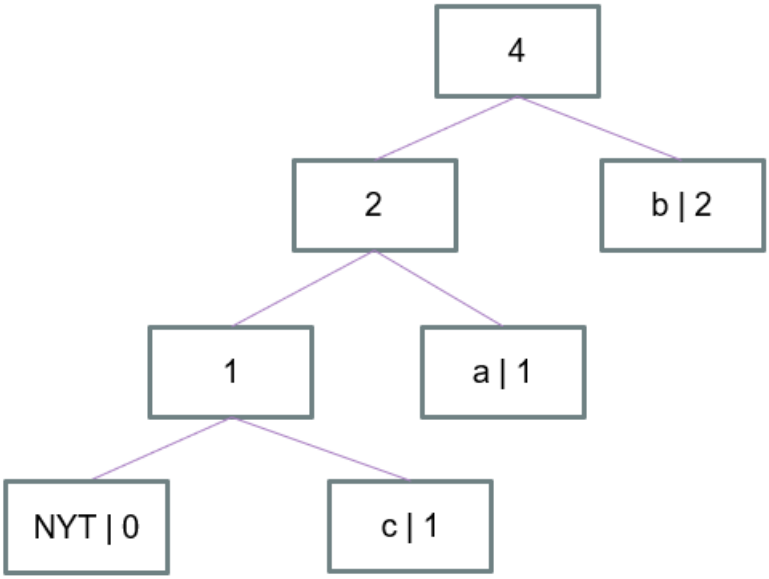
3. **Input = b, output(b) = 001100010**



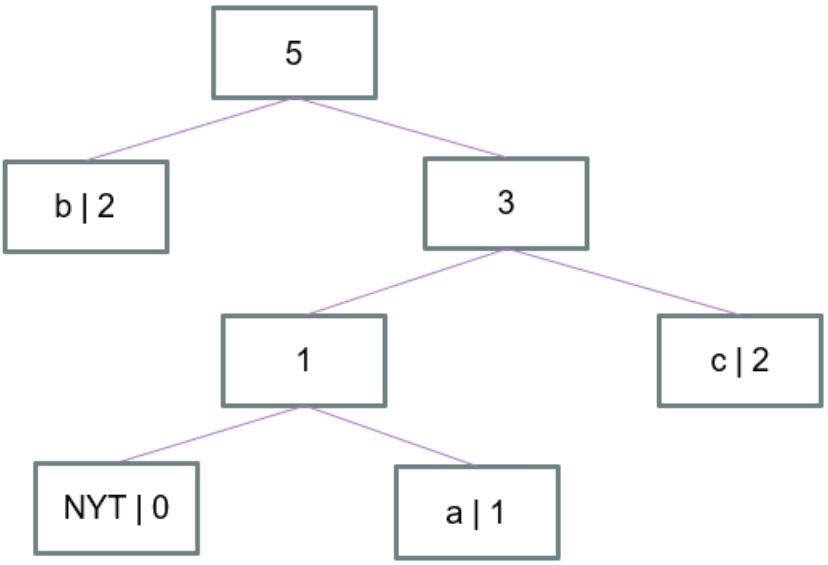
4. **Input = b, output(b) = 01**



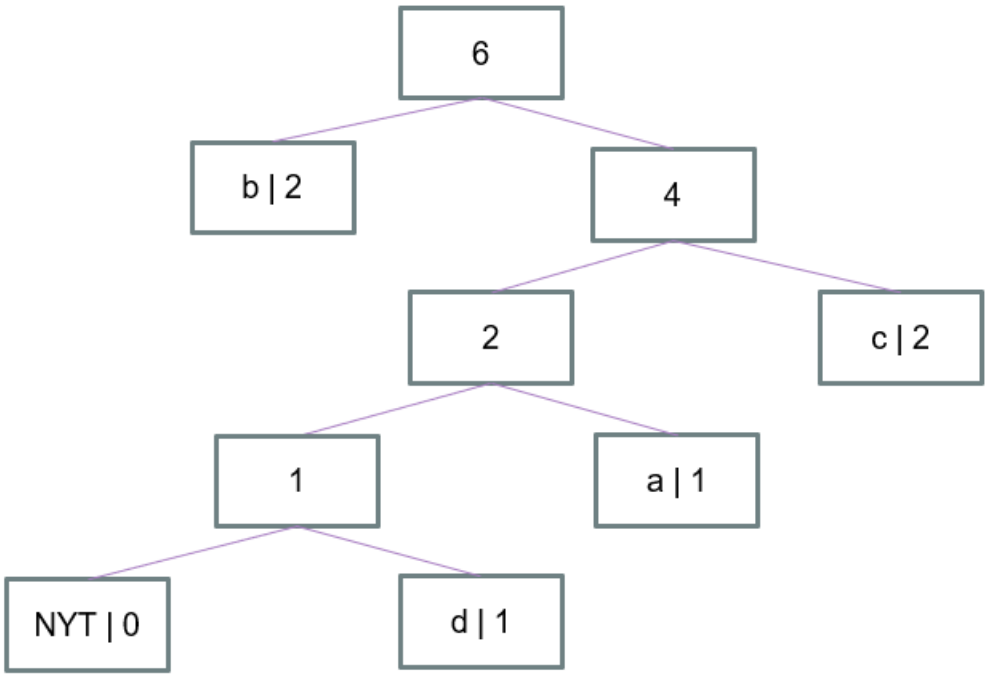
5. **Input = c, output(c) = 0001100011**



6. **Input = c, output(c) = 001**



7. **Input = d, output(d) = 0001100100**



→ $\boldsymbol{output} =$ **01100001001100010010001100011001100010001100100**

2.2.3.2. Thuật toán giải nén

- ***Input và Output***
    - Input: Giá trị đã nén – dữ liệu đã được mã hóa bằng các mã nhị phân.
    - Output: Chuỗi ban đầu trước khi nén.

Thuật toán giải nén Adaptive Huffman Coding bao gồm các bước sau:

Bước 1: Khởi tạo cây Adaptive Huffman với một nút duy nhất chứa ký tự đặc biệt NYT có trọng số bằng 0.

Bước 2: Đọc lần lượt các bit từ dữ liệu nén → giải mã.

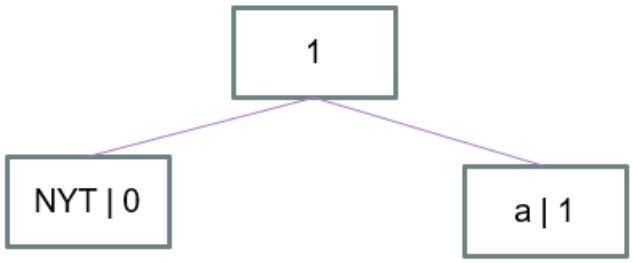
- Ban đầu đọc 8-bit đầu tiên từ từ liệu nén.
- Tiếp theo lấy từng bit b, duyệt trên cây (nếu b = 0→ trái; nếu b = 1 → phải)
    - Nếu đi đến nút lá x → trả về ký tự của nút lá x
    - Nếu đi đến nút NYT → lấy 8-bit tiếp theo để giải mã.

Bước 3: Cập nhật cây Adaptive Huffman giống như quá trình nén.

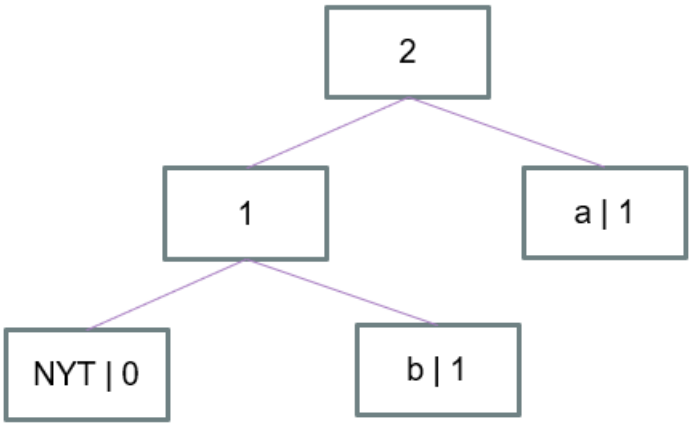
Để dễ tiếp cận các bước của thuật toán giải nén, chúng em có ví dụ đơn giản như sau: cho "01100001001100010010001100011001100010100100" là kết quả của chuỗi "abbccd" được nén ở trên và chạy từng bước thuật toán.

1. **Khởi tạo cây Adaptive Huffman ban đầu**

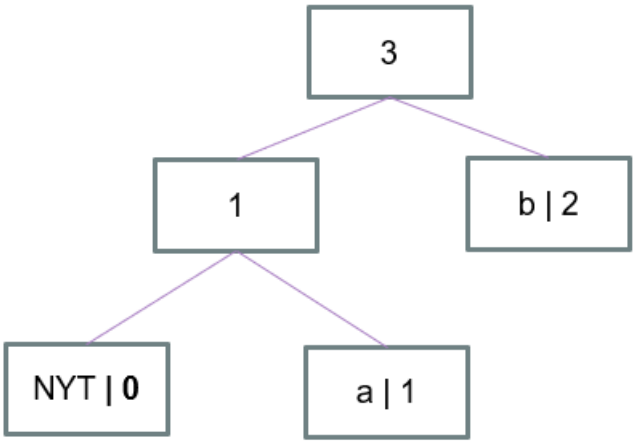
2. **Input = 01100001, output (01100001) = a**



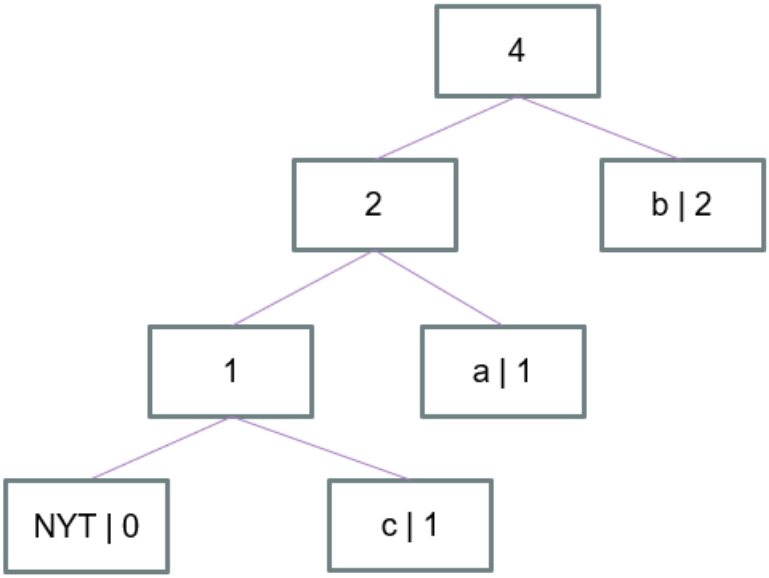
3. **Input = 001100010, output (01100010) = b**



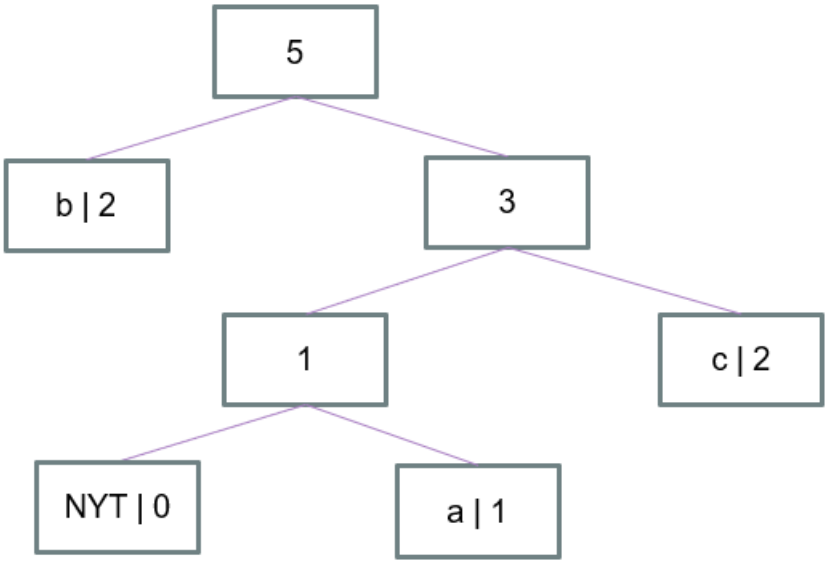
4. **Input = 01, output (01) = b**



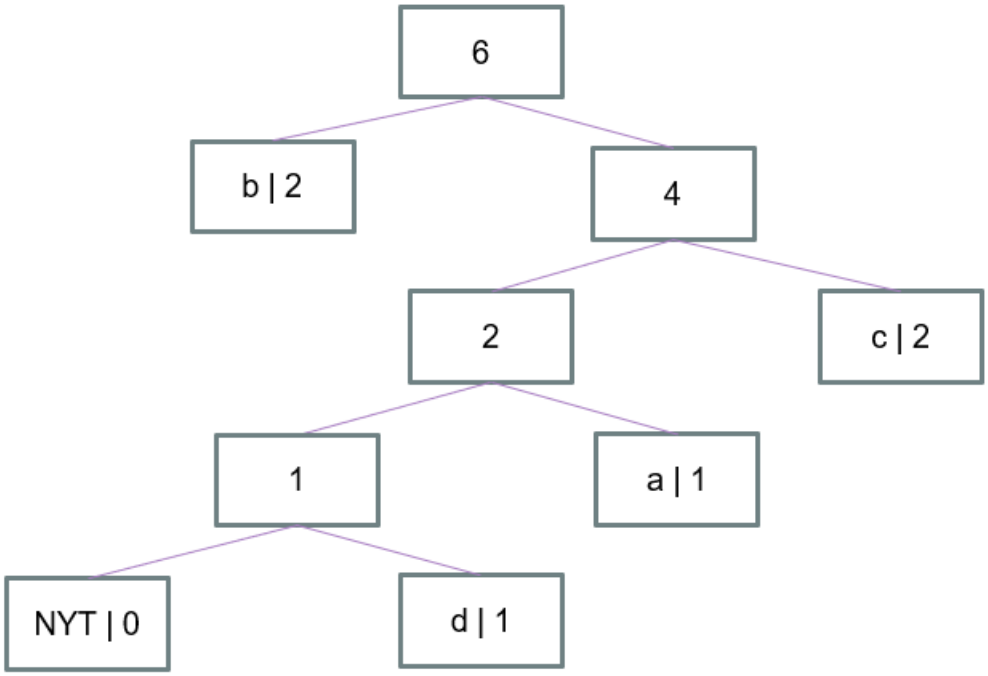
5. **Input = 0001100011, output (01100011) = c**



6. **Input = 011, output (011) = c**



7. Input = 100011100100, output (01100100) = d



$\rightarrow$ ***output = abbccd***

2.2.4. Ưu điểm và hạn chế của Adaptive Huffman

- **Ưu điểm**
  - Khả năng thích ứng: Thuật toán có khả năng thích ứng với các thay đổi trong dữ liệu đầu vào. Nó cung cấp một cách

linh hoạt để xây dựng cây Huffman dựa trên thông tin xuất hiện thực tế trong quá trình nén, từ đó cải thiện hiệu suất nén.
  - Hiệu suất nén cao: Với khả năng xây dựng cây Huffman động, thuật toán Adaptive Huffman Coding có thể tạo ra mã ngắn hơn cho các ký tự xuất hiện thường xuyên, giúp giảm kích thước của dữ liệu nén.
- **Hạn chế**
  - Tính toán phức tạp: So với Huffman Coding truyền thống, thuật toán Adaptive Huffman Coding đòi hỏi tính toán phức tạp hơn do sự thay đổi động của cây Huffman.
  - Không thích hợp cho dữ liệu nhỏ: Trong trường hợp dữ liệu đầu vào có kích thước nhỏ, lợi ích của việc thích ứng cây Huffman có thể không lớn, trong khi chi phí tính toán có thể tăng lên.

Chương 3. ĐÁNH GIÁ

1.1. Độ đo đánh giá

Để so sánh hai thuật toán nén chúng tôi đã sử dụng các độ đo sau: kích thước sau khi nén (Size after compression), tỷ lệ nén (Compression raio), và thời gian nén (Compression time).

Kích thước sau khi nén (Size after compression): Đây là độ đo kích thước tệp sau khi áp dụng thuật toán nén. Giá trị càng nhỏ thể hiện kích thước tệp đã được nén tốt. Để tính toán độ đo này, nhóm chúng tôi sử dụng hàm sys.getsizeof() để lấy kích thước nén sau khi nén.

Tỷ lệ nén (Compression Ratio): Đây là độ đo để đánh giá hiệu suất nén của thuật toán. Tức là kích thước tệp gốc so với kích thước tệp sau nén.

Tỷ lệ nén là một trong các đặc trưng quan trọng của mọi phương pháp nén. Tỷ lệ nén được định nghĩa như sau:

Compression ratio = (kích thước tệp gốc) / (kích thước tệp sau nén)

Thời gian nén (Compression time): Đây là độ đo để đo lường thời gian mà thuật toán mất để nén tệp. Chúng tôi đã sử dụng module time trong Python để đo thời gian và so sánh thời gian nén của các thuật toán khác nhau.

1.2. Kết quả đánh giá

Trong nghiên cứu này, nhóm chúng tôi đã thực hiện đánh giá hiệu xuất nén trên 5 tập dữ liệu với các kích thước khác nhau trong đó 2 tập đầu tiên là văn bản chứa các kí tự, chữ số còn 3,4 và 5 chứa thêm các ký tự đặc biệt hay nói cách khác thì văn bản 3,4 và 5 phức tạp hơn 2 tập đầu. Kết quả của quy trình đánh giá nén được trình bày tương ứng trong bảng 3-1 sau:

Data File Size (bytes)	Size after Compression		Compression ratio		Compression Time(s)	
	Arithmetic	Adaptive Huffman	Arithmetic	Adaptive Huffman	Arithmetic	Adaptive Huffman
3049	2320	2237	1.31	1.36	7	0.182
6049	2320	4402	2.61	1.37	48	0.336
10049	4744	8432	2.12	1.19	281	0.999
15049	4744	12566	3.17	1.20	883	1.432
20049	4744	16730	4.23	1.20	2585	3.568

Bảng 3-1 Kết quả nén văn bản bằng thuật toán Adaptive Huffman và Arithmetic coding

Nhận xét: Dựa vào bảng kết quả thực nghiệm trên có thể thấy rõ về kích thước của tệp sau khi nén và về tỷ lệ nén của thuật toán Arithmetic coding đều tốt hơn so với thuật toán Adaptive Huffman và đặc biệt là khi kích thước của tập dữ liệu càng lớn thì sự chênh lệch về tỷ lệ nén của 2 thuật toán này càng rõ rệt có thể thấy với tập dữ liệu 20049(bytes) thì tỷ lệ nén của thuật toán Arithmetic đạt được tận 4.23 nhưng cùng với tỷ lệ nén cao đó thì thời gian mà thuật toán Arithmetic coding sử dụng để nén tập dữ liệu này là rất lâu, cùng với sự tăng kích thước của tập dữ liệu cần nén thì

thời gian nén của thuật toán Arithmetic coding cũng tăng theo do độ phức tạp tính toán tăng lên.

Có thể thấy một điều đặc biệt trong bảng kết quả trên là kích thước dữ liệu sau khi nén của thuật toán Arithmetic không thay đổi ở 2 tập đầu và 3 tập sau vẫn giống nhau điều này có thể giải thích là như đã nói ở phần thuật toán Arithmetic coding thì output của Arithmetic coding chỉ là một giá trị nén và để cần giải nén thì chúng ta cần có giá trị nén và bảng xác xuất nên kích thước sau khi nén của thuật toán Arithmetic coding được nhóm tính bằng kích thước của giá trị nén và kích thước của bảng xác xuất nên khi số lượng loại kí tự giống nhau thì kích thước dữ liệu lớn hơn thì kích thước của bảng xác xuất cũng không thay đổi chỉ thay đổi giá xác xuất và giá trị nén.

Với thuật toán Adaptive Huffman có thể thấy với các tập dữ liệu nhỏ không quá phức tạp thì Adaptive Huffman vẫn đạt được hiệu xuất khá tốt nhưng đối với các tập dữ liệu lớn hơn và phức tạp hơn thì tỷ lệ nén của thuật toán Adaptive Huffman sẽ bị giảm bớt nhưng vì vẫn duy trì được tốc độ nén tương đối nhanh.

Vì vậy, khi làm việc với các tập dữ liệu không quá lớn và cần hiệu suất nén tốt, thuật toán Arithmetic coding có thể là lựa chọn tốt. Trong khi đó, khi tập dữ liệu lớn và tốc độ nén là yếu tố quan trọng, Adaptive Huffman có thể là sự lựa chọn phù hợp hơn cả.

Do đó, sự lựa chọn giữa hai thuật toán này để tích hợp vào ứng dụng thì còn phụ thuộc vào yêu cầu cụ thể của ứng dụng, có thể là hiệu suất nén cao hoặc tốc độ nén nhanh.

Chương 4. KẾT LUẬN

Trong bài báo cáo này, chúng tôi đã thực hiện nén văn bản sử dụng hai phương pháp là Adaptive Huffman và Arithmetic Coding. Hai phương pháp này đã được nghiên cứu rộng rãi và áp dụng trong lĩnh vực nén dữ liệu.

Phương pháp Arithmetic coding đạt được kết quả tốt và tốn ít dung lượng lưu trữ nhưng đòi hỏi tính toán phức tạp và thời gian nén lâu với dữ liệu lớn. Cùng với đó là thuật toán Adaptive Huffman, phương pháp này là một biến thể của thuật toán Huffman cổ điển. Kết quả cho thấy rằng phương pháp Adaptive Huffman mang lại hiệu xuất nén cao với dữ liệu lớn, đồng thời cho phép thay đổi cấu trúc cây Huffman trong quá trình nén để tăng cường hiệu suất nén.

Tổng kết lại, cả hai phương pháp Adaptive Huffman và Arithmetic Coding đều mang lại kết quả tốt trong việc nén văn bản. Mỗi phương pháp có những ưu điểm và hạn chế riêng, tùy thuộc vào đặc điểm của dữ liệu và mục tiêu nén. Việc chọn phương pháp thích hợp sẽ giúp tối ưu hóa hiệu suất nén và tiết kiệm tài nguyên lưu trữ và băng thông truyền tải. Các phương pháp này có thể được áp dụng trong nhiều lĩnh vực, từ truyền thông dữ liệu đến lưu trữ dữ liệu trên các thiết bị di động,….

Bên cạnh các phương pháp nén văn bản đã được trình bày, chúng tôi cũng có một số hướng phát triển tiềm năng để nâng cao hiệu suất nén và ứng dụng của nén văn bản như sau:

Tối ưu hóa thuật toán: Có thể tiếp tục nghiên cứu và phát triển thuật toán để cải thiện hiệu suất nén. Tìm hiểu thêm về các biến thể của các thuật toán nén có thể dẫn đến việc phát triển các phương pháp nén mới, giảm dung lượng lưu trữ và tăng tốc độ truyền thông.

Kết hợp phương pháp nén: Để đạt được kết quả nén tối ưu, có thể kết hợp các phương pháp nén khác nhau.

Nén văn bản đa ngôn ngữ: Mở rộng phương pháp nén văn bản để hỗ trợ đa ngôn ngữ có thể là một hướng phát triển hứa hẹn. Điều này đòi hỏi nghiên cứu về cách mã hóa và nén các ngôn ngữ khác nhau, cũng như xử lý các đặc điểm ngôn ngữ đặc thù để đạt được hiệu suất nén tối ưu.

Tối ưu hóa truyền tải và lưu trữ: Nén văn bản có thể được ứng dụng trong các lĩnh vực khác nhau như truyền tải dữ liệu và lưu trữ dữ liệu trên các thiết bị di động. Tiếp tục nghiên cứu về cách tối ưu hóa truyền tải và lưu trữ dữ liệu nén có thể mang lại lợi ích lớn về băng thông, dung lượng lưu trữ và thời gian truy cập.

Nghiên cứu về nén dữ liệu thời gian thực: Đối với ứng dụng yêu cầu xử lý dữ liệu thời gian thực, như truyền trực tiếp hoặc truyền tải dữ liệu, việc nén dữ liệu trong thời gian thực trở thành một thách thức. Nghiên cứu về cách nén dữ liệu hiệu quả trong thời gian thực sẽ đóng góp vào các ứng dụng như video trực tiếp, trò chơi trực tuyến và truyền thông đa phương tiện.

TÀI LIỆU THAM KHẢO

Li, Z., Drew, M. S., & Liu, J. (2014). Fundamentals of Multimedia. Springer Science & Business Media.

Wikipedia contributors. (2023). Adaptive Huffman coding. Wikipedia. https://en.wikipedia.org/wiki/Adaptive_Huffman_coding

Oanh-Lem-Linh. (2018b). Adaptive Huffman. dokumen.tips. https://dokumen.tips/documents/adaptive-huffman.html

Gad, A. (2023). Lossless data compression using arithmetic encoding in Python and its applications in deep learning. neptune.ai. https://neptune.ai/blog/lossless-data-compression-using-arithmetic-encoding-in-python-and-its-applications-in-deep-learning

Wikipedia contributors. (2023). Arithmetic coding. Wikipedia. https://en.wikipedia.org/wiki/Arithmetic_coding

Howard, P. G., & Vitter, J. S. (2008). Arithmetic coding for data compression. In Springer eBooks (pp. 65–68). https://doi.org/10.1007/978-0-387-30162-4_34

PHÂN CÔNG CÔNG VIỆC

Sinh viên	Công việc	Mức độ hoàn thành
Trần Văn Lực 20521587	Phân công, hoàn thiện báo cáo, làm slide, code demo, thực nghiệm đánh giá, viết báo cáo chương 1, 3, 4	100%
Nguyễn Văn Hợp 20521358	Viết báo cáo chương 2.2(Adaptive Huffman) và code thuật toán Adaptive Huffman	100%
Ngô Ngọc Sương 20521852	Viết báo cáo chương 2.1(Arithmetic Coding) và code thuật toán Arithmetic Coding	100%