Nhóm 6

Học phần: Công Nghệ Đa Phương Tiện

Lớp: DCCNTT 13.10.16

Đại Học Công Nghệ Đông Á

Các Thành Viên

Họ và Tên	Mã Sinh Viên
Nguyễn Trí Dũng	20223155
Hoàng Ngọc Thành	20223011
Hoàng Văn Nguyên	20223153
Trần Thị Hải Yến	20222897
Trương Hồng Nhuần	20222540

Nội Dung

1. Tổng quan về đề tài
2. Lí thuyết tổng quan về nén dữ liệu
3. Cài đặt mã hóa Huffman
4. Hình ảnh minh họa mã hóa

1

Tổng quan về đề tài

Giới thiệu

01

Một trong những khía cạnh quan trọng của công nghệ hiện nay là bảo mật thông tin. Khi mà thông tin truyền tải qua các mạng truyền thông và internet ngày càng tăng, việc bảo vệ dữ liệu trở nên càng quan trọng. Một trong những công nghệ đóng góp vào lĩnh vực này là kỹ thuật mã hóa.

Kỹ thuật mã hóa là quá trình chuyển đổi thông tin từ dạng thông thường sang dạng mã hóa, làm cho thông tin trở nên không đọc được nếu không có khóa giải mã. Mã hóa Huffman là một phương pháp mã hóa mà bạn có thể tìm thấy rộng rãi trong việc nén dữ liệu.

Thuật toán Huffman được phát minh bởi David A. Huffman vào năm 1952 và đã trở thành một trong những phương pháp nén dữ liệu hiệu quả nhất. Ý tưởng chính của nó là sử dụng các mã độ dài khác nhau cho các ký tự khác nhau trong dữ liệu, sao cho các ký tự xuất hiện thường xuyên sẽ có mã ngắn hơn và ngược lại. Điều này giúp giảm dung lượng của dữ liệu mà không làm mất mát thông tin.

Lí do chọn đề tài

Chúng em đã chọn đề tài kỹ thuật mã hóa Huffman vì nó là một lĩnh vực quan trọng và thú vị trong lĩnh vực khoa học máy tính. Kỹ thuật này không chỉ là một phần quan trọng của lĩnh vực mã hóa thông tin mà còn đề cập đến khá nhiều khái niệm toán học và thuật toán, điều mà chúng em rất quan tâm và mong muốn tìm hiểu sâu hơn.

Ngoài ra, việc nghiên cứu và hiểu rõ về kỹ thuật Huffman cũng mang lại cho chúng em kiến thức vững về cấu trúc dữ liệu và thuật toán,nó cũng giúp chúng em hiểu rõ hơn về nguyên lý hoạt động của các thuật toán mã hóa và nâng cao kiến thức về lĩnh vực kỹ thuật máy tính, hai lĩnh vực quan trọng trong ngành công nghiệp công nghệ thông tin. Việc áp dụng được kiến thức này không chỉ giúp chúng em phát triển kỹ năng lập trình mà còn mở ra cơ hội tham gia vào các dự án lớn và thách thức trong lĩnh vực công nghiệp.

Cuối cùng, chúng em tin rằng việc nghiên cứu và triển khai thành công kỹ thuật mã hóa Huffman sẽ giúp chúng em có cái nhìn sâu sắc hơn về cách các hệ thống thông tin hoạt động và làm thế nào chúng có thể được tối ưu hóa để đáp ứng các yêu cầu ngày càng cao của xã hội hiện đại.

Khái niệm về mã hóa Huffman

Mã hóa Huffman là một phương pháp nén dữ liệu hiệu quả, nơi mỗi ký tự trong chuỗi được biểu diễn bằng một mã nhị phân có độ dài khác nhau. Kết quả của quá trình mã hóa Huffman thường là một bảng mã, trong đó các ký tự thường xuyên xuất hiện được biểu diễn bằng các mã ngắn hơn so với các ký tự ít xuất hiện.

Khi áp dụng mã hóa Huffman, ta thu được dữ liệu đã được nén, giảm kích thước so với dữ liệu gốc. Điều này giúp tiết kiệm không gian lưu trữ và tăng tốc quá trình truyền tải dữ liệu qua mạng. Đồng thời, mã hóa Huffman cũng giúp tối ưu hóa việc lưu trữ dữ liệu trên thiết bị và ổ đĩa, đặc biệt là trong các ứng dụng yêu cầu sự hiệu quả về nguồn lực.

Tóm lại, kết quả của mã hóa Huffman là sự giảm kích thước dữ liệu mà vẫn giữ được thông tin quan trọng, tạo ra một biểu diễn nén hiệu quả và thích hợp cho nhiều ứng dụng lưu trữ và truyền tải dữ liệu.

2

Lí thuyết tổng quan về nén dữ liệu

Khái niệm về nén dữ liệu

Nén dữ liệu (tiếng Anh: Data compression) là việc chuyển định dạng thông tin sử dụng ít bit hơn cách thể hiện ở dữ liệu gốc. Tùy theo dữ liệu có bị thay đổi trước và sau khi giải nén không, người ta chia nén thành hai loại: Nguyên vẹn (lossless) và bị mất dữ liệu (lossy). Nén mất dữ liệu giảm số lượng bit bằng cách xác định các thông tin không cần thiết và loại bỏ chúng.

Nén dữ liệu là cần thiết vì giảm được nguồn tài nguyên cũng như dung lượng lưu trữ hay băng thông đường truyền. Tuy nhiên, vì dữ liệu nén cần được giải nén nên sẽ đòi hỏi nhiều phần cứng và xử lý.

Tỉ lệ

Tỉ lệ nén là một đo lường cho sự giảm kích thước của một tệp tin hoặc dữ liệu so với kích thước ban đầu. Nó được tính bằng cách so sánh kích thước trước và sau khi áp dụng kỹ thuật nén.

$$Tỉ\ lệ\ nén\ =\ (1 - \frac{Kích\ thước\ dữ\ liệu\ sau\ khi\ nén}{Kích\ thức\ dữ\ liệu\ ban\ đầu}) * 100$$

Tỉ lệ nén thường được biểu diễn dưới dạng một phần trăm hoặc một tỷ lệ số, và càng cao thì dữ liệu đã nén càng hiệu quả.

Một số khái niệm cơ bản

Trong kỹ thuật nén dữ liệu, độ dư thừa (redundancy) là sự lặp lại hoặc dư thừa của thông tin trong dữ liệu. Đối với nén dữ liệu, mục tiêu chính là giảm độ dư thừa để giảm kích thước của dữ liệu mà vẫn giữ được thông tin cần thiết. Có bốn loại độ dư thừa chính trong kỹ thuật nén dữ liệu:

a, Sự lặp lại của những ký tự

b, Sự phân bố của những ký tự

c, Độ dư thừa vị trí

d, Những mẫu sử dụng mật độ cao

Một số khái niệm cơ bản

a, Sự lặp lại của những kí tự

Trong một nguồn dữ liệu, thường có những kí tự và chuỗi kí tự lặp lại nhiều lần liên tiếp nhau. Khi đó, nguồn dữ liệu có thể được mã háo một cách cô đọng hơn bằng cách thay đổi các kí tự đó bằng mã của chúng và số kí tự lặp lại.

b, Sự phân bố của các kí tự

Xét trong một dãy kí tự, ta thường thấy có một số kí tự xuất hiện với tần suất cao hơn so với các kí tự khác. Như vậy, ta có thể giảm bớt lượng dữ liệu bằng cách mã hóa những kí tự xuất hiện thường xuyên với từ mã ngắn, những kí tự ít xuất hiện hơn sẽ được mã hóa bằng những từ mã dài hơn. Kiểu dư thừa này đặc biệt phù hợp với phương pháp mã hóa Huffman.

Một số khái niệm cơ bản

c, Độ dư thừa vị trí

Có nhiều trường hợp, dữ liệu trong một nguồn dữ liệu có sự phụ thuộc lẫn nhau, do đó nếu biết được kí hiệu xuất hiện tại một vị trí nào đó, ta có thể phỏng đón trước một cách hợp lý sự xuất hiện của các kí hiệu khác ở những vị trí khác nhau.

Ví dụ, ảnh biểu diễn trong một lưới 2 chiều, một số điểm ở hàng dọc lại xuất hiện trong cùng vị trí đó ở các hàng khác nhau. Như vậy, thay vìu lưu trữ dữ liệu ta chỉ lưu lại vị trí hàng và cột. Phương pháp nén khai thác kiểu dư thừa này gọi là phương pháp mã hóa dự đoán.

Một số khái niệm cơ bản

d, Những mẫu sử dụng mật độ cao

Thông thường, trong các văn bản dạng text, sự tuần tự của những kí tự bào đó sẽ tái xuất hiện với tần suất tương đối cao. Vì vậy, có thể biểu diễn bằng dãy bit ngắn hơn.

Để dánh giá một thuật toán nén có hiệu quả hay không, người ta sẽ dựa vào cách mà thuật toán xử lý các kiểu dư thừa như trên. Thực tế cho thấy rằng, hầu hết các ký thuật nén đều không đủ mềm dẻo để xử lý tất cả các kiểu dư thừa. Mỗi chiến lược nén áp dụng thường chỉ cứng nhắc cho từng kiểu số liệu mà thôi.

Độ dư thừa số liệu có thể định lượng bằng toán học. Với L1, L2 là hai đại lượng số liệu cùng được dùng để biểu diễn một lượng tin cho trước thì độ dư số liệu tương đối RD của tập số liệu thứu nhất so với tập số liệu thứ hai là:

$$R_D = 1 - \frac{1}{\frac{L_1}{L_2}}$$

Trong đó:

L1 / L2 được gọi là tỉ lệ nén.

RD là độ dư số liệu tương đối.

Hai dạng nén

a, Nén không tổn hao (Lossless Compression)

Nén không tổn hao còn gọi là nén chính xác hay nén không mất thông tin. Đây là phương pháp nén mà sau khi giải nén ta thu được một bản sao chính xác của dữ liệu gốc. Phương pháp nén này thường được áp dụng đối với các nguồn số liệu mà nội dung thông tin cần được bảo toàn như các văn bản dạng text, các bảng tính hay là cơ sở dữ liệu...

Dạng nén mà em dùng trong bài này là dạng nén không tổn hao.

b, Nén tổn hao (Lossy Compression)

Nén tổn hao còn được gọi là nén có mất mát thông tin. Kỹ thuật nén này chấp nhận mất mát một lượng thông tin nhất định để thu được hiệu suất nén cao hơn, do vậy sau khi giải nén ta không thu được dữ liệu gốc.

Nén hao tổn thường được áp dụng cho các tập tin hình ảnh hay âm thanh được số hóa. Bởi vì đối với các tập tin thuộc loại này thì việc mất mát một ít thông tin là điều có thể chấp nhận được.

Định nghĩa mã hóa

Mã hóa là quá trình biến đổi thông tin từ dạng có thể đọc được sang dạng không thể đọc được nếu không có khóa hoặc thuật toán để giải mã. Mục đích của mã hóa là bảo vệ tính bí mật, toàn vẹn và khả dụng của thông tin khi truyền, lưu trữ hoặc xử lý. Mã hóa có thể được thực hiện bằng nhiều cách khác nhau, tùy thuộc vào mức độ an toàn, tốc độ và chi phí mong muốn. Một số loại mã hóa phổ biến hiện nay là:

Mã hóa cổ điển: là loại mã hóa đơn giản nhất, không cần khóa bảo mật, chỉ cần người gửi và người nhận biết thuật toán mã hóa. Ví dụ: mật mã Caesar, mật mã Vigenère, mật mã Playfair, v.v.

Mã hóa đối xứng: là loại mã hóa sử dụng cùng một khóa cho cả quá trình mã hóa và giải mã. Khóa này phải được chia sẻ bí mật giữa người gửi và người nhận. Ví dụ: mật mã DES, mật mã AES, mật mã RC4, v.v.

Mã hóa bất đối xứng: là loại mã hóa sử dụng hai khóa khác nhau cho quá trình mã hóa và giải mã. Một khóa được gọi là khóa công khai, có thể được phổ biến cho mọi người, và một khóa được gọi là khóa bí mật, chỉ được giữ riêng cho chủ sở hữu. Ví dụ: mật mã RSA, mật mã ECC, mật mã ElGamal, v.v.

Một số khái niệm cơ bản

a, Chiều dài từ mã

Chiều dài từ mã hóa (đôi khi được gọi là mã dài) thường được sử dụng trong ngành khoa học máy tính và lĩnh vực mã hóa hóa. Đây là một khái niệm quan trọng trong kích thước xác định của dữ liệu được mã hóa.

Trong bối cảnh này, chiều dài của mã hóa thường được sử dụng để mô tả số lượng bitcas hoặc ký tự trong một mã hóa chuỗi. Chiều dài từ mã là số kí hiệu của bộ mã dùng để mã hóa cho mã đó.

Một số khái niệm cơ bản

b, Trọng lượng từ mã

Trọng lượng từ mã trong mã hóa Huffman là tổng số bit cần thiết để mã hóa một ký tự cụ thể trong một chuỗi. Nó được xác định bằng chiều dài của mã Huffman được gán cho ký tự đó. Trong mã hóa Huffman, mỗi ký tự được gán một mã nhị phân duy nhất.

Mã này được xây dựng dựa trên tần suất xuất hiện của các ký tự trong chuỗi. Các ký tự xuất hiện thường xuyên hơn sẽ được gán mã ngắn hơn. Các ký tự xuất hiện ít hơn sẽ được gán mã dài hơn. Trọng lượng từ mã của một ký tự bằng số lượng bit cần thiết để mã hóa ký tự đó.

Tổng trọng lượng từ mã của tất cả các ký tự trong chuỗi bằng độ dài trung bình của mã Huffman. Độ dài trung bình của mã Huffman càng nhỏ thì mã hóa càng hiệu quả. Trọng lượng từ mã là tổng số các kí hiệu khác 0 của từ mã đó.
VD: Từ mã 1011010 có trọng lượng là 4.

Một số khái niệm cơ bản

c, Khoảng cách mã

Trong mã hóa Huffman, khoảng cách mã (code distance) là số lượng bit ít nhất giữa hai mã Huffman khác nhau. Khoảng cách mã quan trọng vì nó ảnh hưởng đến hiệu suất chung của mã Huffman.

Khoảng cách mã lý tưởng của một mã Huffman là 2, vì điều này đảm bảo rằng mỗi ký tự được mã hóa bằng một mã duy nhất và không có hai mã nào chồng lấn lên nhau. Tuy nhiên, trong thực tế, không phải lúc nào cũng có thể đạt được khoảng cách mã là 2. Khi không thể đạt được khoảng cách mã là 2, thì khoảng cách mã càng lớn thì mã Huffman càng hiệu quả hơn.

Có hai yếu tố chính ảnh hưởng đến khoảng cách mã của một mã Huffman:

Tần suất xuất hiện của các ký tự: Các ký tự xuất hiện thường xuyên hơn sẽ có mã ngắn hơn, trong khi các ký tự xuất hiện ít thường xuyên hơn sẽ có mã dài hơn. Điều này dẫn đến sự chồng chéo giữa các mã của các ký tự khác nhau và giảm khoảng cách mã.

Số lượng ký tự: Số lượng ký tự trong một bảng chữ cái cũng ảnh hưởng đến khoảng cách mã. Với số lượng ký tự càng lớn, thì khoảng cách mã càng khó đạt được.

Các phương pháp biểu diễn mã

a, Phương pháp liệu kê

Liệt kê trong một bảng những thông tin của nguồn và kèm theo là các từ mã tương ứng. Ưu điểm của phương pháp này là rõ ràng, đơn giản nhưng không phù hợp với những bộ mã lớn.

VD: Nguồn tin X = {a, b, c, d}. Các lớp tin được mã hóa như sau:

Tin	a	b	c	d
Mã	10	01	110	001

Các phương pháp biểu diễn mã

b, Phương pháp đồ hình kết cấu

Phương pháp này biểu diễn mà bằng một cây mã rút gọn bao gồm các nút và các nhánh có hướng. Một vòng kín bắt đầu tại nút gốc , đi theo các nhánh theo chiều mũi tên, qua các nút trung gian và kết thúc tại nút gốc sẽ biểu diễn cho 1 từ mã. Thứ tự các nhánh trên dường đi chính là thư tự giá trị các kí hiệu.

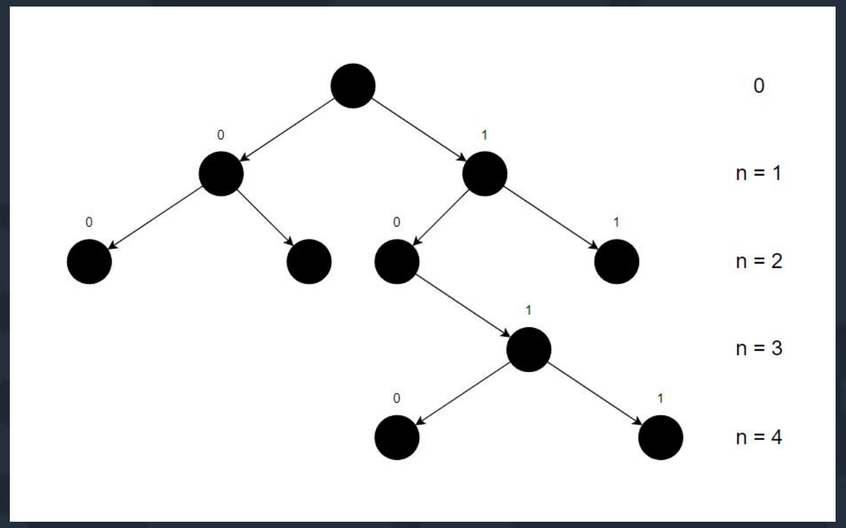
Các phương pháp biểu diễn mã

c, Phương pháp cây

Cây mã được biểu diễn bao gồm gốc và các nhánh. Trong cây có chứa các nút. Nút gốc chính là gốc của cây (mức 0). Nút là nằm tận cùng của nhánh. Trừ nút gốc và các nút lá ra, các nút còn lại là các nút nhánh. Từ một nút nhánh có thể phát đi nhiều nhất m nhánh (ứng với cơ số m của mã). Mỗi nhánh biểu diễn cho 1 từ mã. Từ mã đó thứ tự các trị kí hiệu đi từ gốc, qua từ nút nhánh và dừng lại ở nút lá tương ứng của nhánh.

Dựa vào cây mã, chúng ta có thể nhận biết mã đã cho là mã đều (các nút lá có cùng bậc), hay không đều, mã đầy hay vơi. Mã là đầy khi mọi nút nhánh bậc trước các nút là đều có m nhánh.

VD: Cho bộ mã 00, 01, 11, 1010, 1011. Cây mã biển diễn của bộ mã này là:



Mã thống kê tối ưu

Như đã nói, tiêu chuẩn của mã thông kê tối ưu là chiều dài trung bình từ mã tối thiểu. Do xác suất xuất hiện của các từ trong nguồn tin là khác nhau nên việc dùng các từ mã ngắn để mã hóa cho các từ có xác suất xuất hiện cao và ngược lại, dùng các từ mã dài để mã hóa cho các từ có xác suất xuất hiện thấp sẽ làm cho số kí hiệu cần thiết để mã hóa nguồn tin giảm đi. Nguyên tắc cơ bản của mã thông kê tối ưu là dựa trên cơ sở độ dài từ mã n (tỉ lệ nghịch với xác suất xuất hiện p), tức là các tin có xác suất xuất hiện thấp sẽ được biểu diễn bằng các từ mã dài và ngược lại.

Mã Shannon - Fano

Phương pháp mã hóa đầu tiên được nhiều người biết đến vào cuối những năm 1940 là phương pháp mã hóa Shannon-Fano. Phương pháp này được hai nhà nghiên cứu Claude Shannon và Robert Fano đưa ra gần như đồng thời. Phương pháp mã hóa này chưa dựa trên tần suất hiện của mỗi ký tự trong nguồn số liệu vậy mà được mô tả chi tiết trong sách của Claude Shannon và Warren Weaver "The Mathematical Theory of Communication". Từ đây một số thuật toán khác đã xuất hiện có tính chất tương tự. Kỹ thuật mã hóa này dựa trên tần suất xuất hiện của mỗi ký tự trong nguồn số liệu cần được mã hóa. Từ bảng chứa các tần suất đó, bảng mã sẽ được xây dựng vào các tính chất quan trọng sau:

- Các mã khác nhau có các bit biểu diễn khác nhau.
- Kí tự cơ bản xuất hiện càng cao thì mã càng ngắn (ít bit) và ngược lại.
- Các mã có dạng dãy bit khác nhau.

Mã Shannon - Fano

Mã sẽ được xây dựng dựa trên cấu trúc cây nhị phân, dựa vào Thuật toán xây dựng mã Shannon-Fano:

Vào: Bảng tần số xuất hiện của từng các kí tự có mặt trong nguồn số liệu (Bảng đã được sắp xếp theo thứ tự tăng dần hoặc giảm dần của tần số).

Ra: Cây nhị phân biển diễn mã.

Bước 1. Tách bảng thành hai bảng con sao cho hiệu giữa tổng các tần số trong mỗi bảng con là nhỏ nhất.

Bước 2. Bảng con phía trên được gán giá trị nhị phân 0, bảng con phía dưới được gán trị nhị phân 1.

Bước 3. Tiếp tục thực hiện tuần tự hai bước 1 và 2 cho mỗi bảng con được tách ra cho đến khi các bảng thành phần không thể phân chia được nữa

Nguyên tắc chính là sử dụng phương pháp đệ qui để xây dựng cây mã.

Mã số học

Như đã nói, độ dài từ mã của mã Shannon-Fano phải là một số nguyên các bit. Người ta đã cải tiến nhược điểm này bằng cách đưa ra một loại mã khác, đó là mã số học. Phương pháp mã hóa số học hoàn hảo hơn các phương pháp mã hóa khác ở chỗ nó không tạo ra một từ mã dơn lẻ cho mỗi kí hiệu mà nó chỉ tạo ra một từ mã duy nhất cho toàn bộ nguồn số liệu. Nghĩa là một kí hiệu có thể được mã hóa bằng 3.5 bit.

Nguyên tắc chính của phương pháp mã hóa này là mã hóa toàn bộ lượng số liệu thành một số. Mỗi kí tự / xâu kí tự của luồng nhập sẽ được biến đổi thành một số thực có giá trị thuộc nửa khoảng [0;1). Việc biến đổi này tuân theo ánh xạ 1-1.

Đối với phương pháp này, trước hết, chúng ta cần lập bảng thống kê tần số xuất hiện của các kí tự. Sau đó gán cho mỗi kí tự một khoảng biên thiên và gọi là hàng của kí tự đó.

Mã hóa huffman

a, Mô hình nguồn số liệu

Như ta đã biết, Entropy của nguồn số liệu phụ thuộc vào xác suất, trong khi đó, xác suất lại phụ thuộc vào mô hình. Do đó, xác suất sẽ thay đổi nếu như chúng ta thay đổi mô hình và Entropy cũng biến đổi theo. Như vậy, có thể thấy rằng hiệu quả nén phụ thuộc rất nhiều vào mô hình.

Nhìn chung, quá trình nén không thể hoàn hảo được thực hiện dựa vào một trong hai kiểu mô hình khác nhau: mô hình thông kê (Statistical) và mô hình từ điển (Dictionary-based). Nén theo mô hình thông kê sẽ mã hoá mỗi lúc một kí hiệu dựa vào tần suất xuất hiện của nó. Nén theo mô hình từ điển sẽ mã hoá mỗi lúc một chuỗi kí hiệu chỉ bằng một từ mã. Như vậy, vai trò của mô hình là vô cùng quan trọng. Một mó hinh tốt sẽ cho hiệu quả nén cao và ngược lại.

Mã hóa huffman

b, Mô hình thống kê

Dạng đơn giản nhất của mô hình này, đúng như tên gọi của nó, là thống kê các khối số liệu điển hình nào đó để có được một bảng tĩnh liệt kê các giá trị tần suất. Dựa vào bảng này, một cây mã tĩnh được xây dựng sẵn và lưu trữ để có thể sử dụng nhiều lần. Một mô hình như thế được gọi là mô hình thống kê tĩnh (Static statistical model).

Mã hóa huffman

c, Mô hình từ điển

Đặc điểm chung của các mô hình thông kê là mã hóa (và giải mã) một lúc một kí hiệu. Còn các mô hình từ điển thì tạo mã cho một cụm hoặc loạt kí hiệu. Nguyên tắc của chúng là tạo một ảnh xạ từ một chuỗi kí hiệu thành mã sao cho kích thước của mã nhỏ hơn kích thước của chuỗi kí hiệu đó. Khi mã hóa, dữ liệu được đọc vào và thuật toán tìm xem có nhóm kí hiệu tương tự nào xuất hiện trong từ điển hay không. Nếu có, nó sẽ xuất ra một mã ảnh xạ đến nhóm kí hiệu đó. Dữ liệu vào càng trùng hợp với các nhóm kí hiệu trong từ điển hoặc kich thước nhóm ki hiệu được ảnh xạ càng lớn thì hiểu quả nén càng cao. Ở đây, vai trò của mô hình hóa là cực kỳ quan trọng, công việc mã hoá chỉ đóng vai trò thứ yếu.

Thực tế cho thấy, so với các kỹ thuật nén sử dụng mô hình thống kê, các kỹ thuật nén áp dụng mô hình từ điển cho một hiệu quả cao hơn nhiều, cả về tỉ số nén, tốc độ né và giải nén. Đó là lý do chúng được sử dụng phổ biến hiện nay.

3

Cài đặt mã hóa Huffman

Bước 1

Bước 1. Đọc dữ liệu đầu vào: Sử dụng C++ để đọc dữ liệu từ một tệp tin ngoài. (lấy dữ liệu từ ngoài)

```
string inputFile = "CNDPT_Nhom6.txt";

 // Bước 1: Đọc dữ liệu từ tệp tin đầu vào
 ifstream inFile(inputFile, ios::binary | ios::ate);
 streampos fileSize = inFile.tellg();
  inFile.seekg(0, ios::beg);
 string data((unsigned int)fileSize, ' ');
 inFile.read(&data[0], fileSize);
 inFile.close();
```

Bước 2

Bước 2. Xây dựng bảng thống kê tần xuất suất hiện của các ký tự: Sử dụng một map để lưu tần suất xuất hiện của từng ký tự trong dữ liệu.

```
// Bước 2: Lập bảng thống kê
 map<char, int> freqTable;
 for (char c : data) {
  freqTable[c]++;
 }
```

Bước 3

- Bước 3. Xây dựng cây huffman từ bang thống kê: Sử dụng một hàng đợi ưu tiện (priority_queue) để xây dựng cây huffman từ các nút lá có tần suất thấp nhất.

```
// Định nghĩa một nút trong cây Huffman
struct Node {
 char data;
 int freq;
 Node* left, *right;

 Node(char data, int freq) : data(data), freq(freq),
left(nullptr), right(nullptr) {}
};
// So sánh để sử dụng trong priority_queue
struct Compare {
 bool operator()(Node* left, Node* right) {
  return left->freq > right->freq;
 }
};
```

Bước 3


```
// Tạo cây Huffman từ bảng thống kê
Node* buildHuffmanTree(map<char, int>& freqTable) {
 priority_queue<Node*, vector<Node*>, Compare> pq;
 for (auto& pair : freqTable) {
  pq.push(new Node(pair.first, pair.second));
 }
 while (pq.size() > 1) {
  Node* left = pq.top(); pq.pop();
  Node* right = pq.top(); pq.pop();
  Node* merged = new Node('$', left->freq + right->freq);
  merged->left = left;
  merged->right = right;
  pq.push(merged);
 }
 return pq.top();
}
```

Bước 4

Bước 4. Xây dựng bảng mã huffman

Sử dụng đệ quy để duyệt cây huffman và xây dựng bảng mã cho từng ký tự.

```
// Tạo bảng mã Huffman
void buildHuffmanCodes(Node* root, string code, map<char, string>&
huffmanCodes) {
 if (root == nullptr) return;

 if (root->data != '$') {
  huffmanCodes[root->data] = code;
 }

 buildHuffmanCodes(root->left, code + "0", huffmanCodes);
 buildHuffmanCodes(root->right, code + "1", huffmanCodes);
}
```

Bước 5

- Bước 5. Mã hóa dữ liệu và xuất kết quả:
- Sử dụng bảng mã huffman để mã hóa dữ liệu và xuất kết quả ra màn hình.

```
// Mã hóa dữ liệu và trả về kết quả
string encodeData(string data, map<char, string>& huffmanCodes) {
 string encodedData = "";
 for (char c : data) {
  encodedData += huffmanCodes[c];
 }

 return encodedData;
}
```

Bước 6

- Bước 6. Tính toán kích thước dữ liệu:
- Tính kích thước dữ liệu ban đầu và kích thước dữ liệu đã mã hóa để tính tỉ lệ nén.

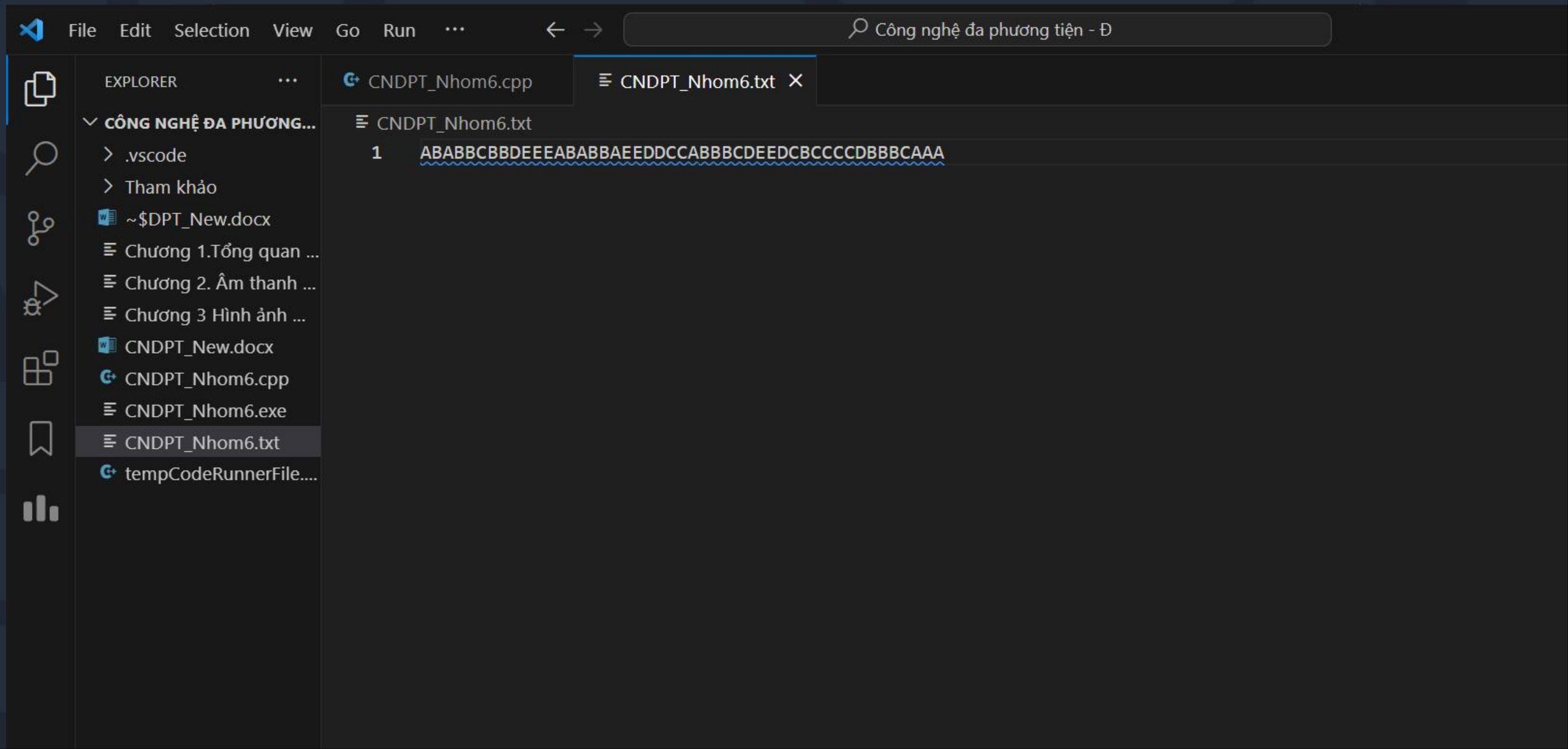
```
// Bước 6: Tính toán input bit và output byte
int inputBits = fileSize * 8;
int outputBytes = (encodedData.length()); // làm tròn lên khi cần
thiết

// Bước 7: Tính tỷ lệ nén
double compressionRatio = ((double)(inputBits-outputBytes) /
inputBits*100;
// In thông tin
cout << "Input byte: " << inputBits << " bytes" << endl;
cout << "Ouput byte: " << outputBytes << " bytes" << endl;
cout << "Ti le nen: " << compressionRatio << endl;
```

4

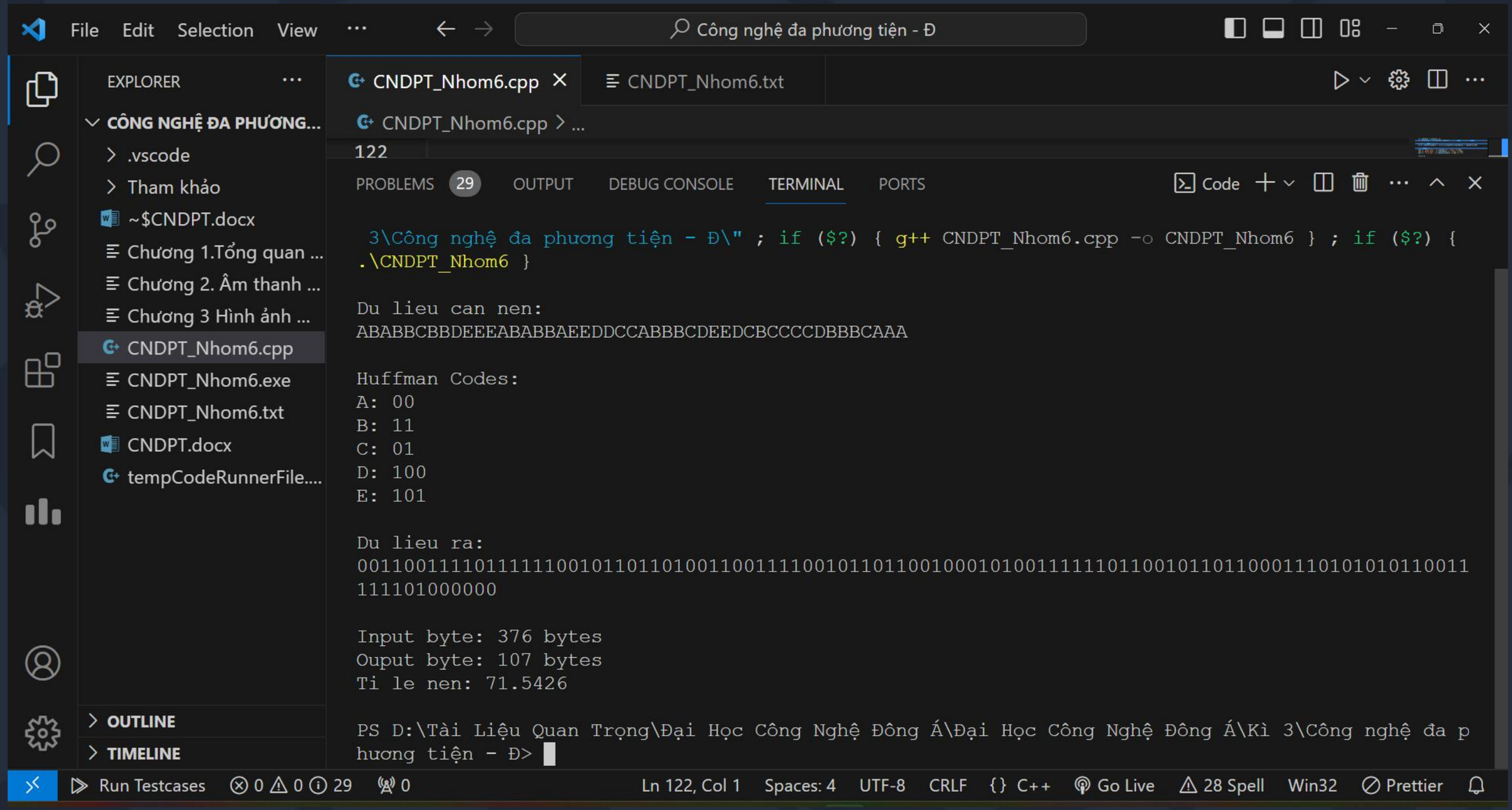
Hình ảnh minh họa mã hóa

Logo

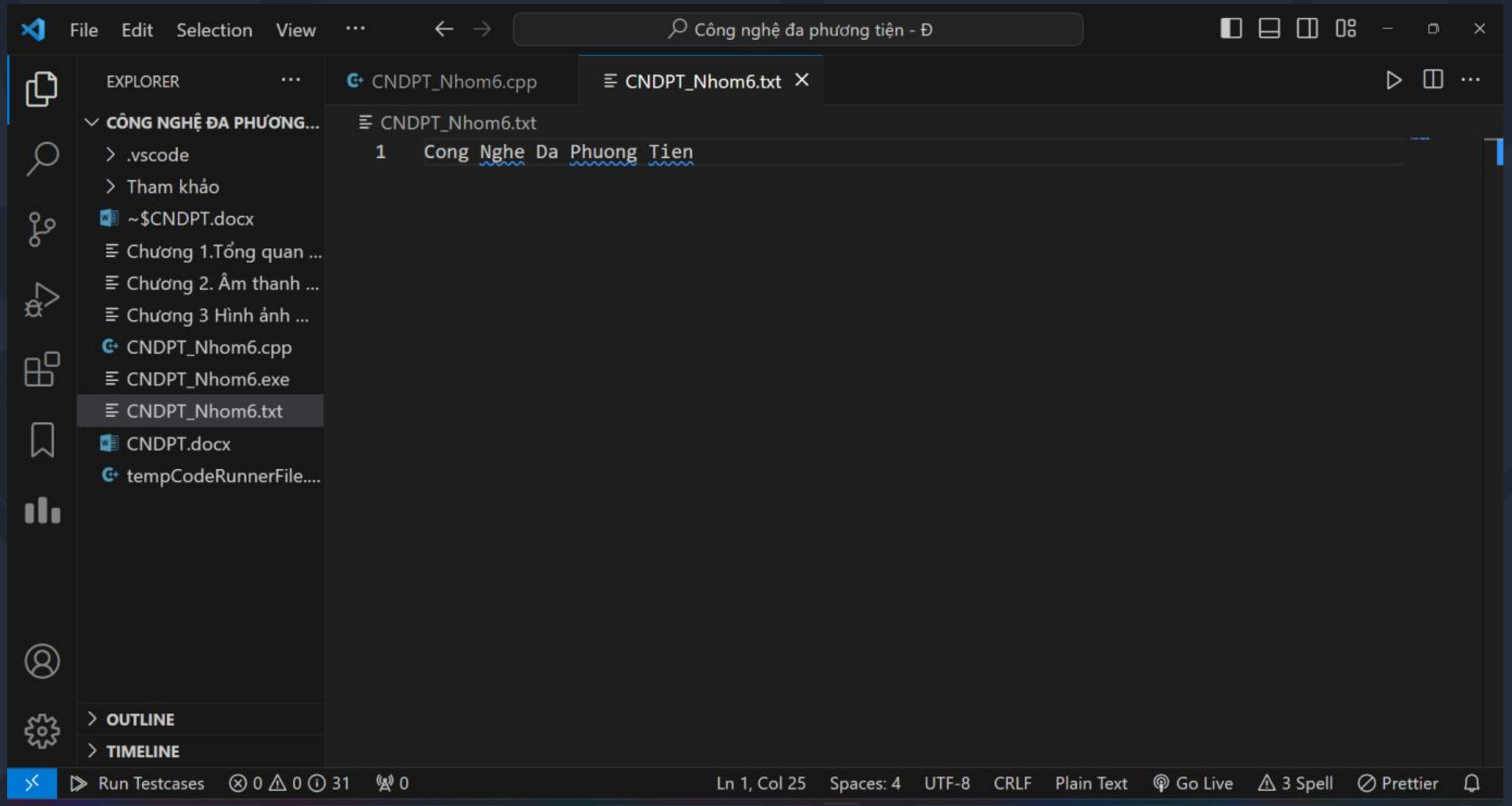
Hình ảnh minh họa mã hóa



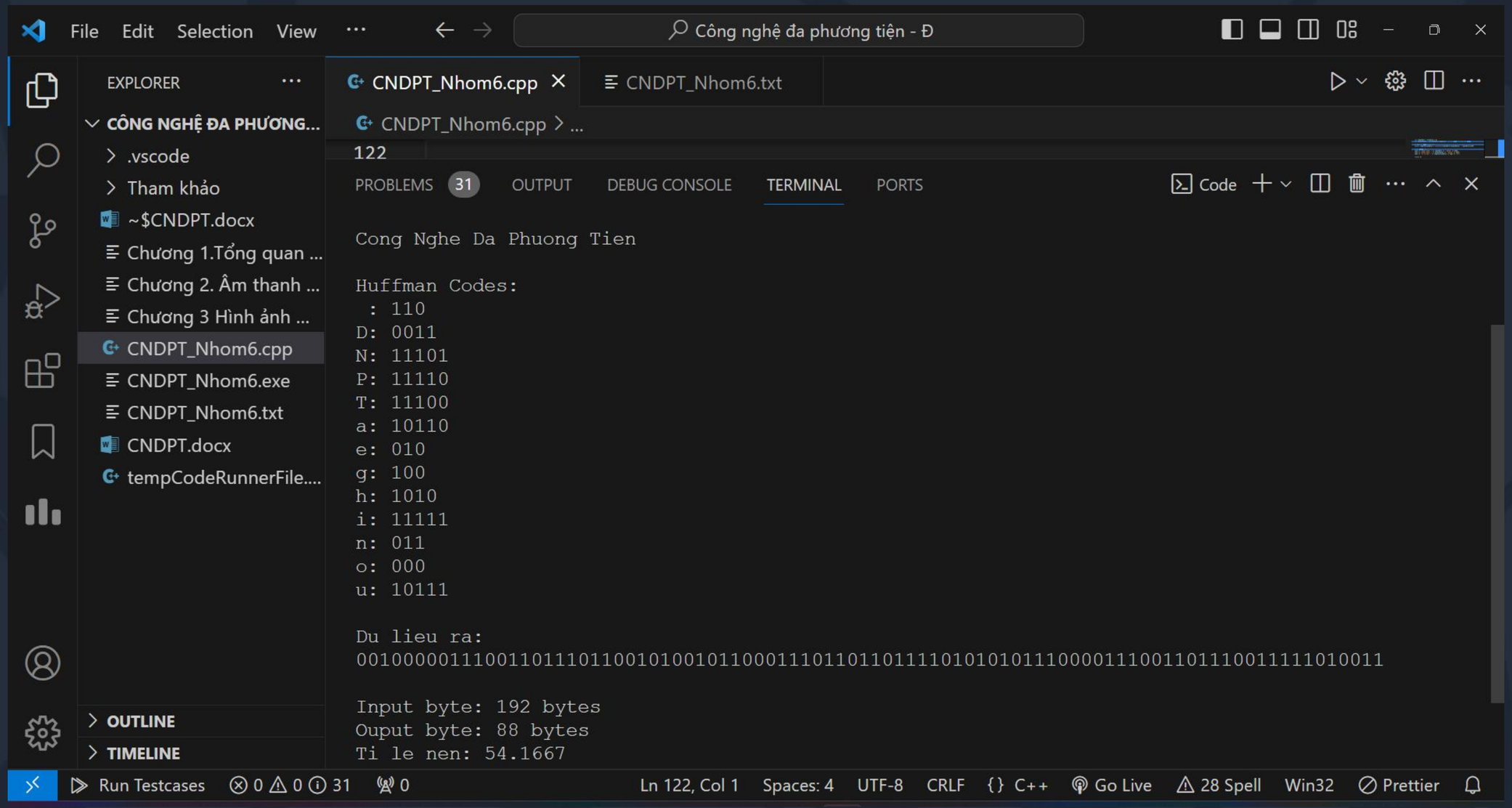
Hình ảnh minh họa mã hóa



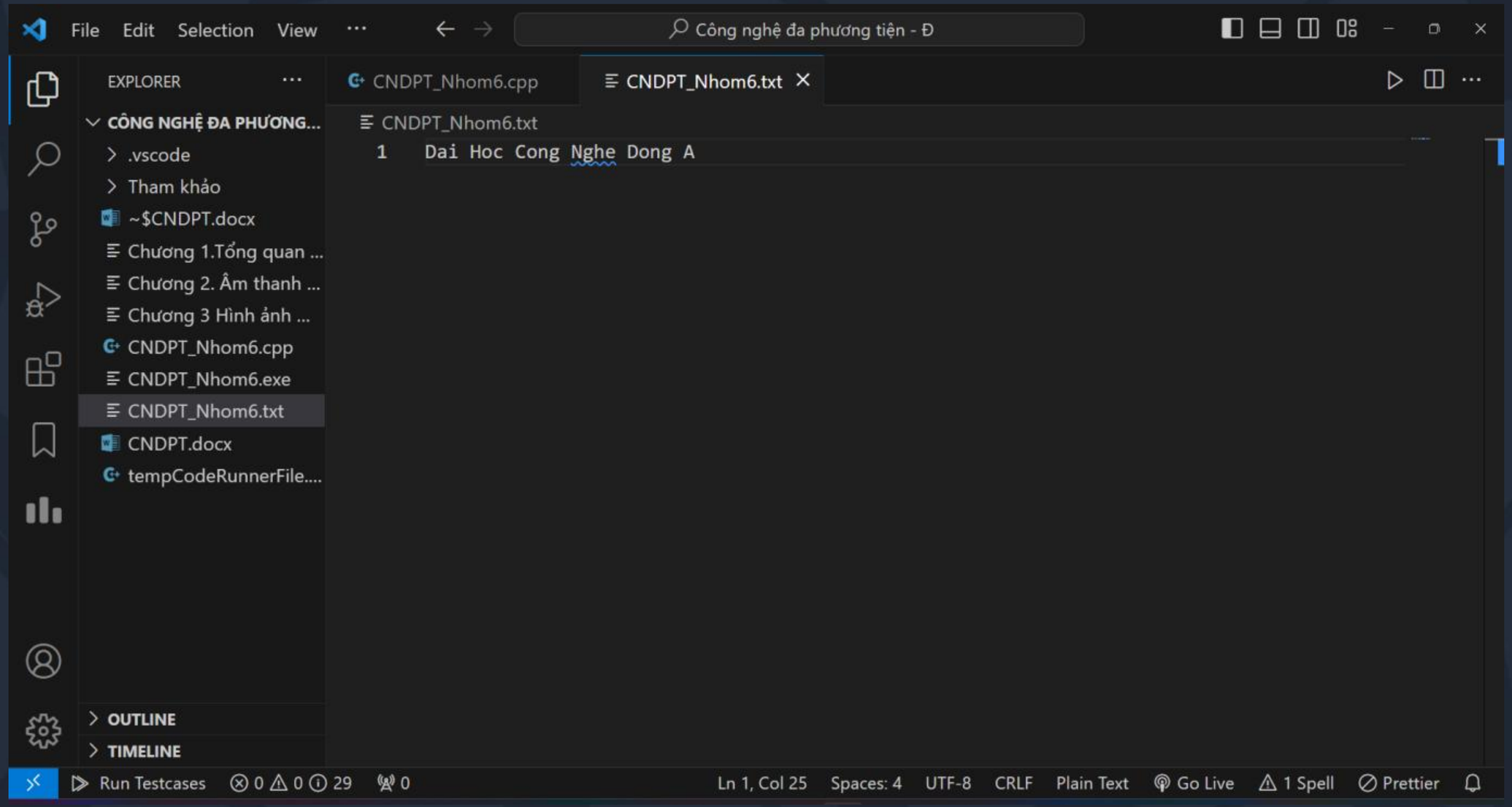
Hình ảnh minh họa mã hóa



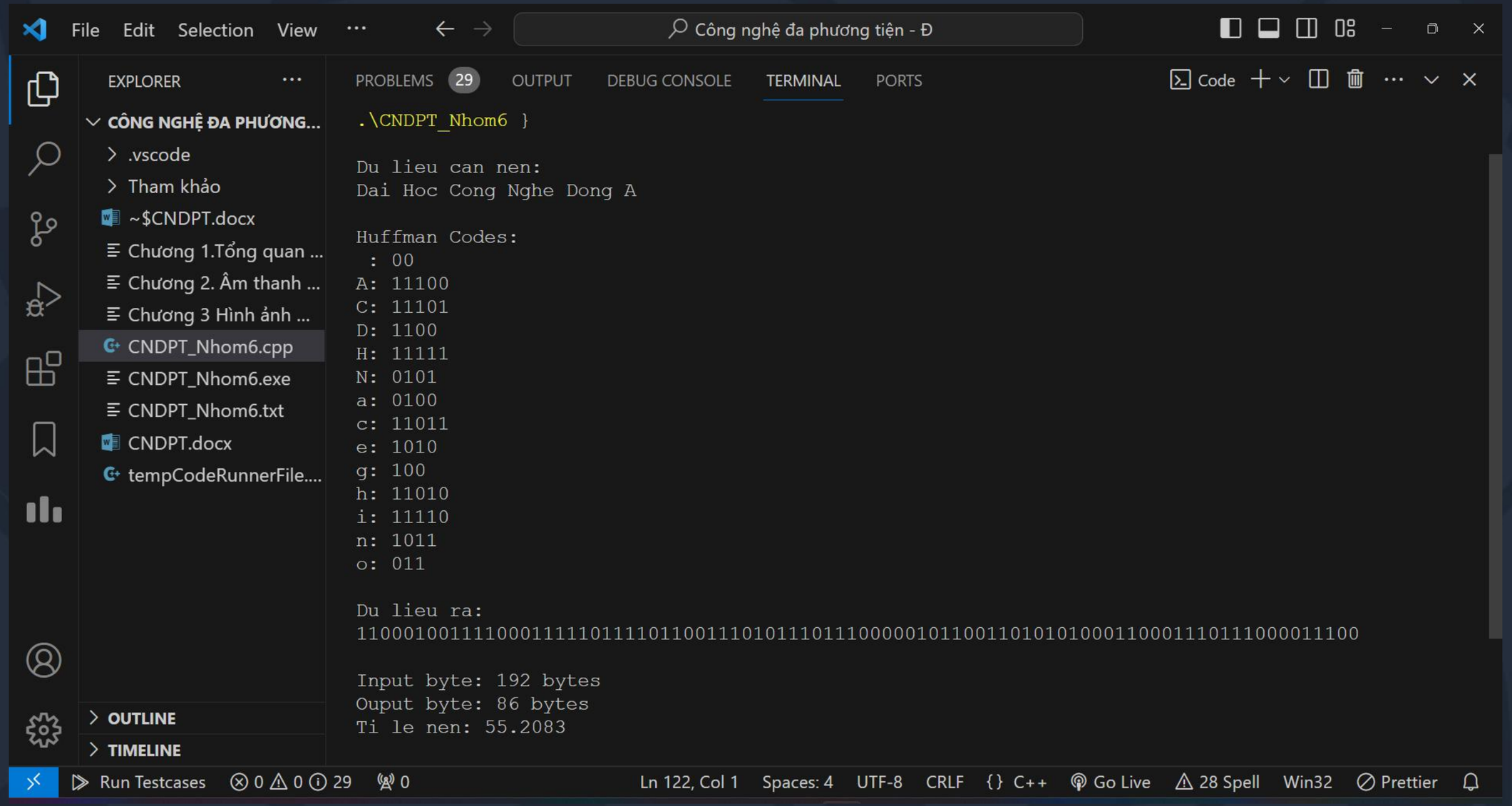
Hình ảnh minh họa mã hóa



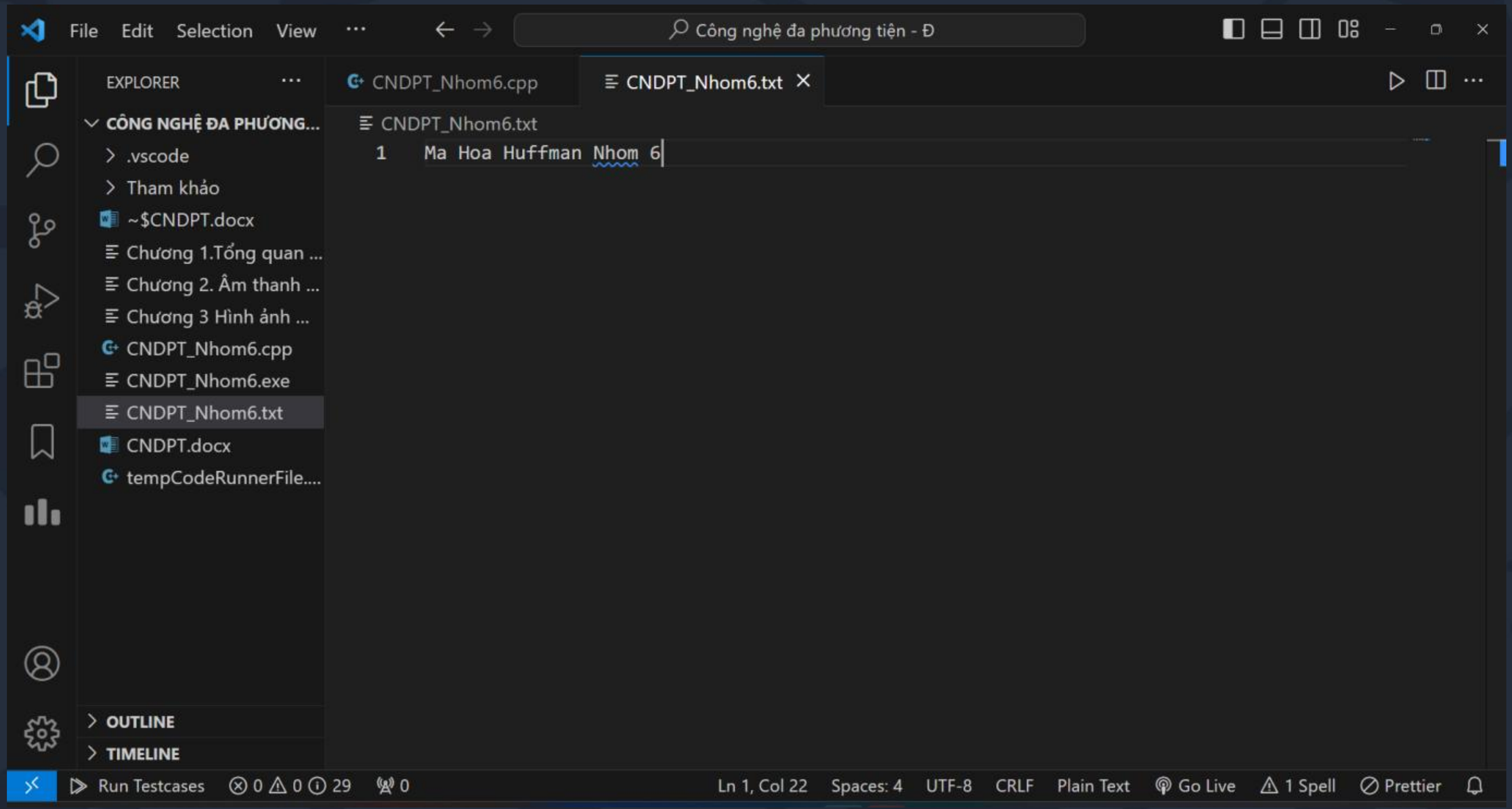
Hình ảnh minh họa mã hóa



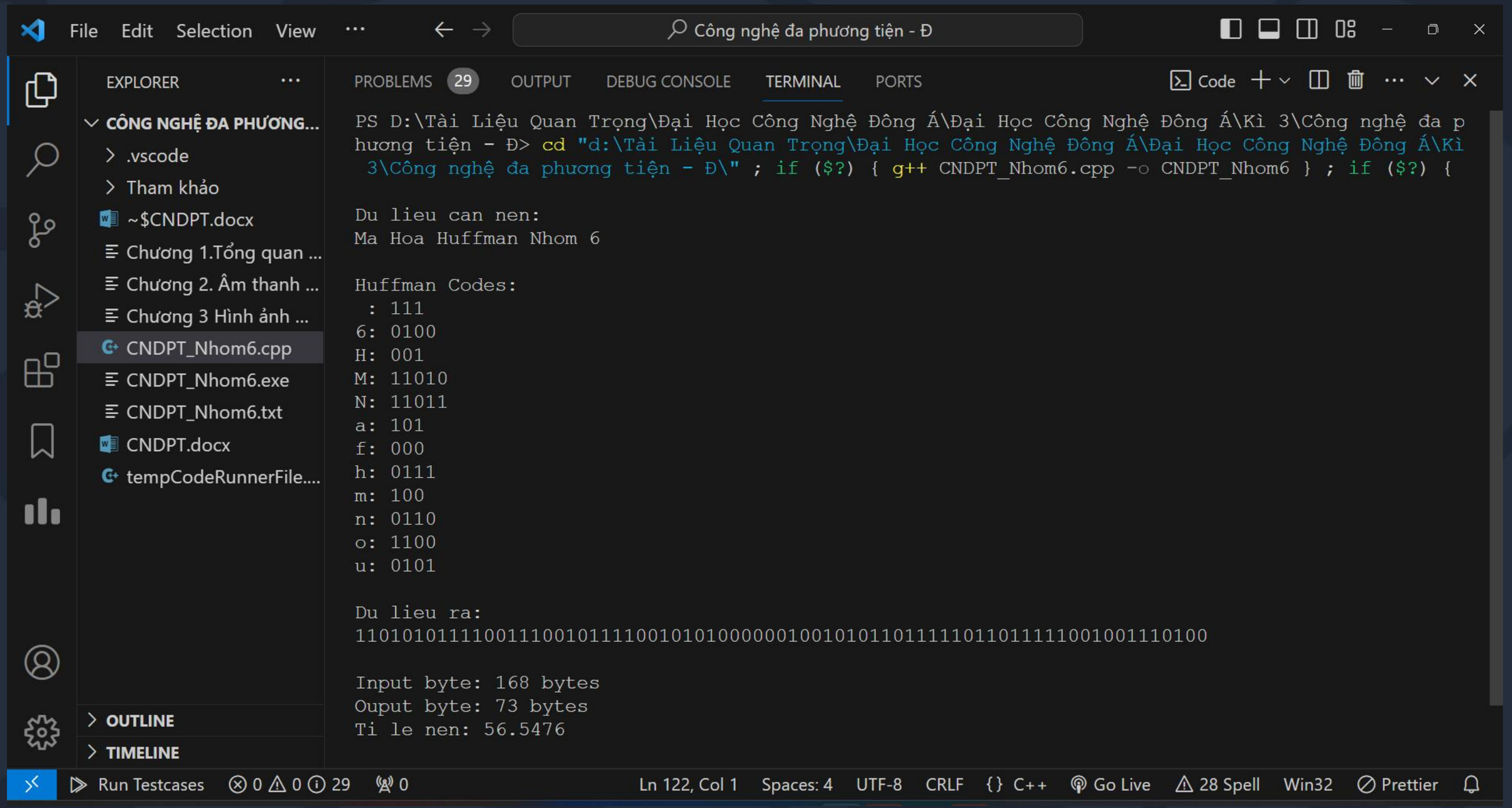
Hình ảnh minh họa mã hóa



Hình ảnh minh họa mã hóa



Hình ảnh minh họa mã hóa



Nhóm 6

Học phần: Nguyên Lí Hệ Điều Hành

THANKS!