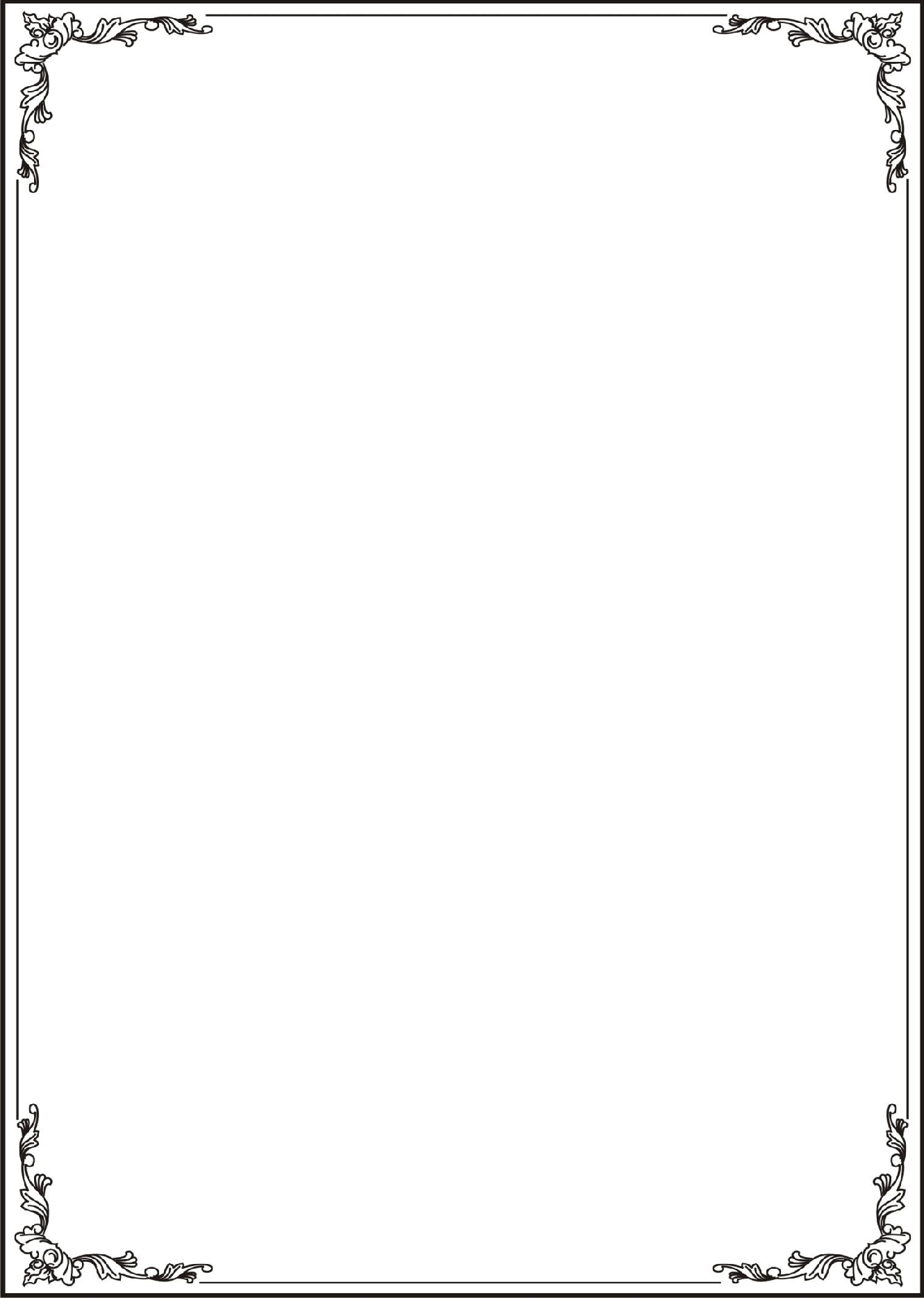
**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP.HCM**



**KHOA CÔNG NGHỆ THÔNG TIN**



**ĐỒ ÁN**

**MÔN HỌC: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT**

***ĐỀ TÀI*: DYNAMIC HUFFMAN CODING**

**GVHD: Th.s Huỳnh Xuân Phụng**

**Mã Lớp:** DASA230179\_22\_1\_08

**Họ và Tên MSSV**

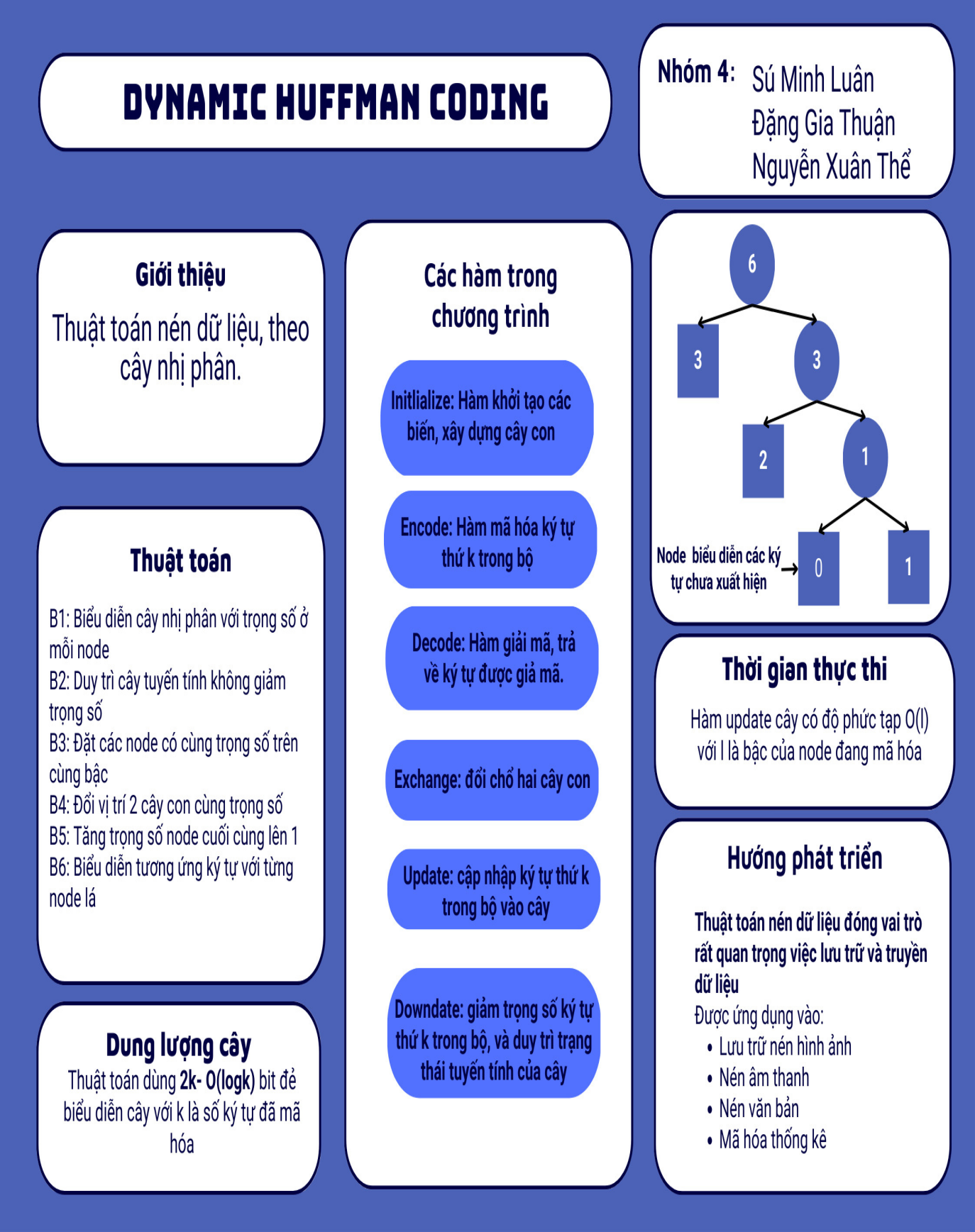
Đặng Gia Thuận 21110665

Nguyễn Xuân Thể 21110927

Sú Minh Luân 21110900

**Thành Phố Hồ Chí Minh, tháng 12 năm 2022**

**\* Thiết kế Poster**



**MỞ ĐẦU**

**1. Lý do chọn đề tài**

Cùng với sự phát triển của hệ thống thông tin và dữ liệu trên toàn thế giới, nhu cầu về truyền và xử lí dữ liệu ngày càng được quan tâm. Nguồn dữ liệu có dung lượng vô cùng lớn, vì thế cần biện pháp để giảm tải tạm thời, truyền dữ liệu một cách dễ dàng hơn. Trong các phương thức để nén thì Dynamic Huffman Coding thường được sử dụng hiện nay. Vì thế nhóm đã quyết định chọn đề tài này để phân tích, làm rõ các ứng dụng của chúng

**2. Mục tiêu nghiên cứu**

Xây dựng được cây Huffman, mã hóa được thông tin đầu vào, giải mã được chuỗi nhị phân

**3. Phương pháp nghiên cứu**

Phương pháp phân loại, hệ thống hóa kiến thức: thông qua bài báo được giáo viên cung cấp để tiến hành phân loại các ý chính của bài, từng bước để xây dựng được cây nhị phân

Phương pháp thu thập, phân tích tài liệu: thông qua các tài liệu trên internet, sách tham khảo để tìm hiểu thêm về nội dung nghiên cứu

**CÁC NỘI DUNG**

**1. Phân tích thuật toán**

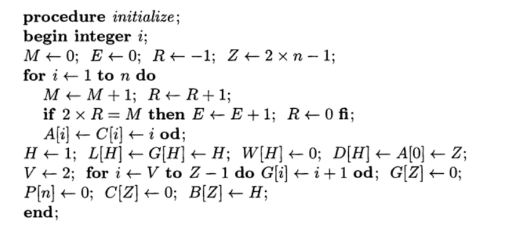
**- Finiteness (Tính hữu hạn):** Chương trình thực hiện sau hữu hạn bước

Thuật toán kết thúc sau 6 bước.

**- Definiteness (Tính xác định):** Chức năng từng bước

Thực hiện 6 bước sau:

**B1**: Bước khởi tạo, biểu diễn cây nhị phân với các trọng số trong mỗi nút;



Phân tích tốc độ của chương trình khởi tạo: tốc độ bước(chương trình khởi tạo) là O(n).

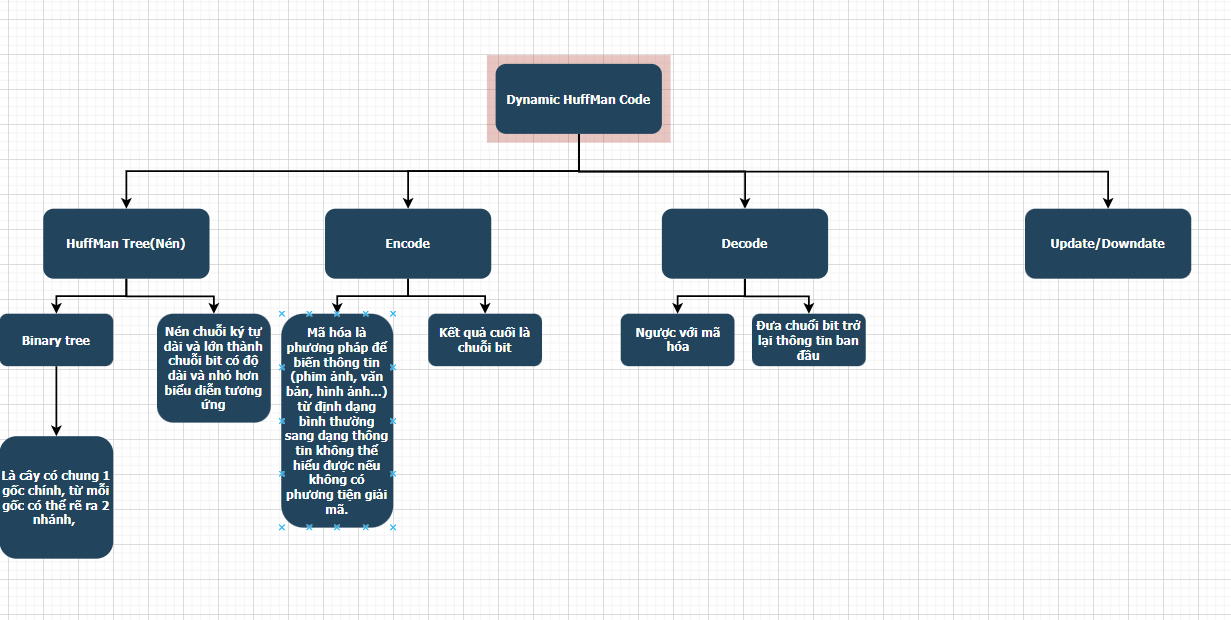
**B2:** Duy trì một danh sách tuyến tính các nút, theo thứ tự không giảm theo trọng số;

**B3**: Cho một nút trong danh sách tuyến tính này, thuộc một khối các nút có cùng trọng số, hãy tìm nút cuối cùng trong khối của nó;

**B4:** Đổi chổ hai cây con có cùng trọng số;

**B5:** Tạo nút gốc mới bằng cách cộng 2 nút có trọng số thất nhấp vào nhau;

**B6:** Biểu diễn sự tương ứng giữa các ký tự và các nút bên ngoài là cây huffman: Bằng cách mã hóa cây huffman sang nhị phân



**- Input (Đầu vào):** Dữ liệu đưa vào là 1 dãy ký tự n phần tử.

Bảng n ký tự A = {a1, a2, …, an}.

Tập các trọng số (tần số xuất hiện của chữ cái tương ứng) tương ứng W = {w1, w2, …, wn}, Ví dụ:

wi = weight(ai), 1 ≤ i ≤ n.

**- Output (Đầu ra):** Dữ liệu đầu ra là 1 dãy nhị phân biểu diễn cho dãy n ký tự đưa vào.

Bộ mã C (A, W) = {c1, c2, …, cn}, là tập hợp các từ mã (Nhị phân), trong đó ci là từ mã của ai, 1 ≤ i ≤ n.

***\* Phân tích chi tiết Code***

- Các biến sử dụng trong bài code

+ n: số ký tự trong hệ biểu diễn

+ S[n]: biểu diễn bit mã hóa cho ký tự đang mã hóa

+ M = số ký tự có trọng số bằng không, chưa xuất hiện

+ E: bit biểu diễn mã hóa

+ R: số bit thừa trong công thức M = 2^E + R

+ P[n]: con trỏ trỏ đến node cha; P[j] là cha của node 2j-1 và 2j; trừ trường hợp P[n] = 0;

+ C[2\*n]: con trỏ trỏ đến node con, nếu node k không là node lá và có node 2j-1 và 2j là node con thì C[k] = j; Còn nếu node k là node là thì C[k] biểu diễn ký tự aC[k].

+ A[n+1]: Biểu diễn ký tự alphabet. Nếu ak là ký tự thứ j có trọng số 0; thì A[k] = j <=M; mặt khác A[k] biểu diễn số của node lá tương ứng ký tự ak, trừ trường hợp A[0] mang một ý nghĩa khác.

+ B[2\*n]: trỏ đến các node

+ W[2\*n]: trọng số của node tương ứng. block k có trọng số W[k];

+ L[2\*n]: con trỏ trái các node. Block gần nhất có trọng số nhỏ hơn block k là block L[k]; trừ trường hợp block k có trọng số nhỏ nhất thì L[k] là block có trọng số lớn nhất

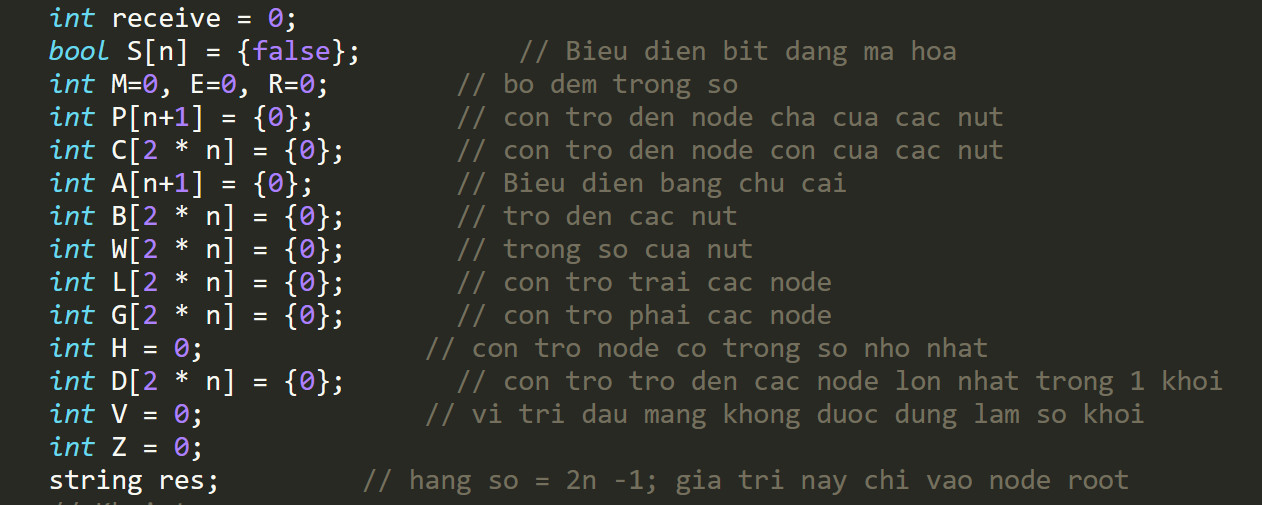
+ G[2\*n]: con trỏ phải node. Block gần nhất có trọng số lớn hơn block k là block G[k]; trừ trường hợp block k có trọng số lớn nhất thì G[k] là block có trọng số nhỏ nhất.

+ H: trỏ đến node có trọng số nhỏ nhất

+ D[2\*n]: con trỏ đến node lớn nhất trong 1 khối

+ V: Con trỏ trỏ đầu danh sách vị trí mà hiện không sử dụng

+ Z: mang giá trị 2n-1; giá trị này trỏ đến node gốc.

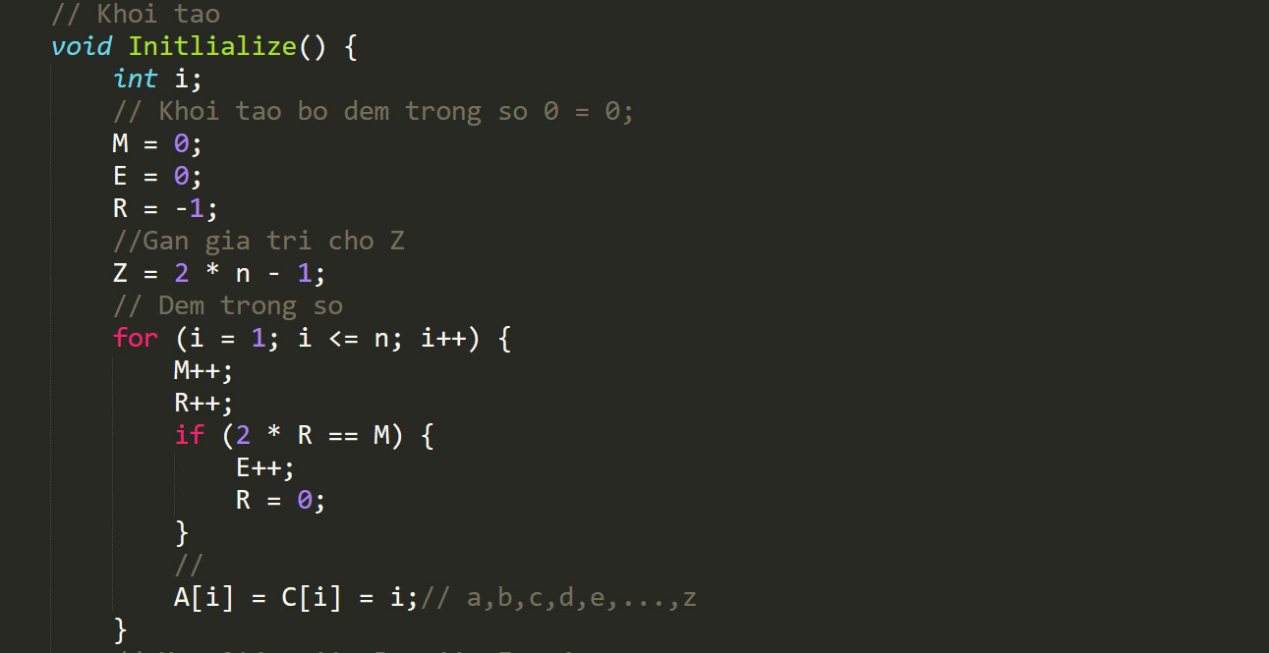


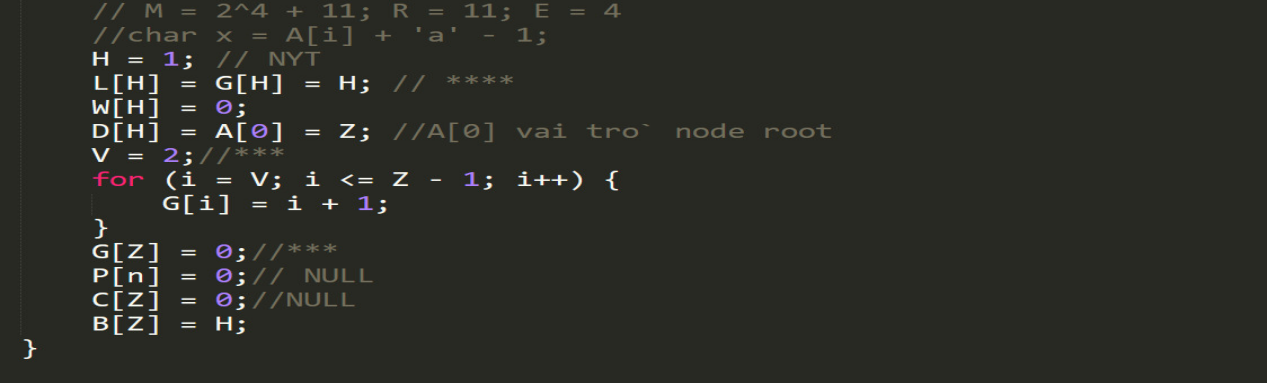
***- Khởi tạo các giá trị của biến***

+ Khởi tạo bộ đếm trọng số

+ Gán trọng số nhỏ nhất H = 1;

+ Tạo các biến liên quan H





***- Hàm Encode:*** tiến hành mã hóa chuỗi kí tự cho trước thành chuỗi nhị phân

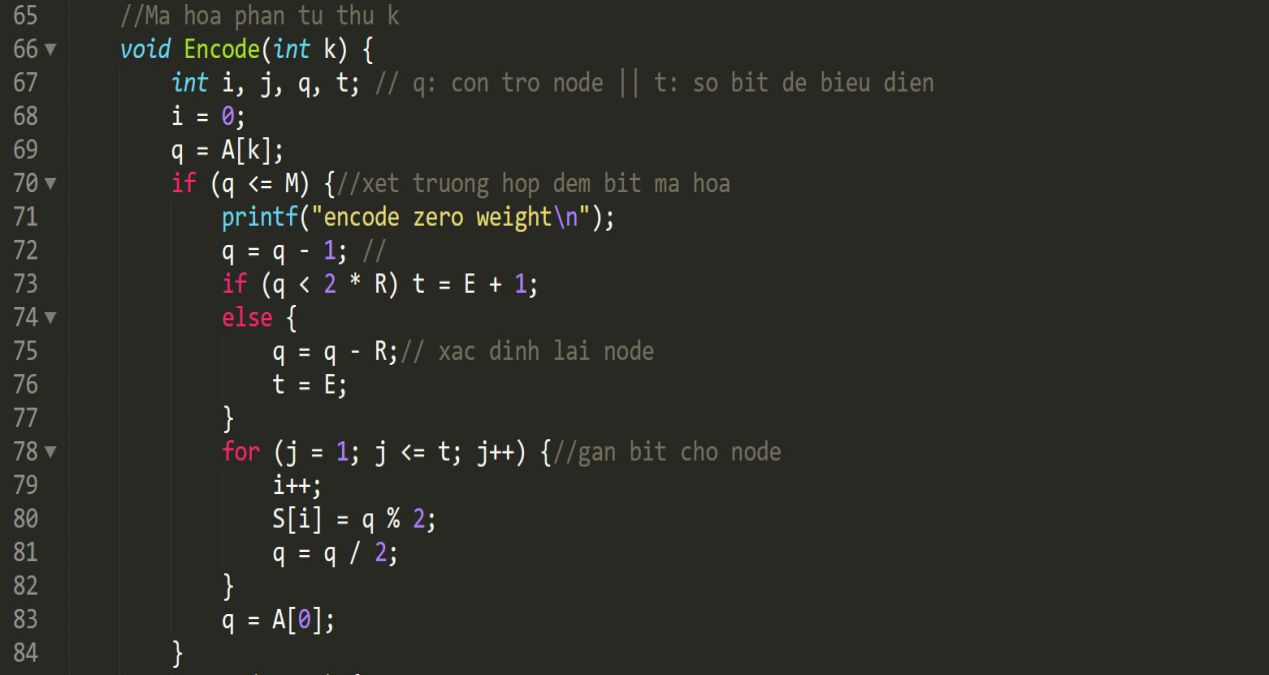
Nếu node q là node có trọng số 0 (chưa xuất hiện) thực hiện quá trình mã hóa trọng số 0

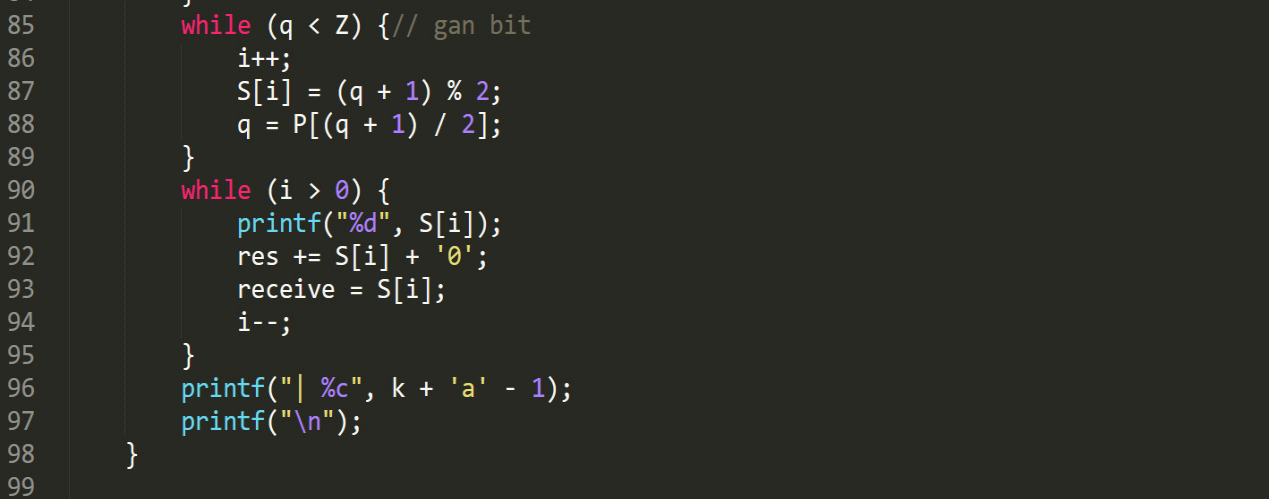
+ In ra dòng “encode zero weight”;

+ Xác định số bit mã hóa cho node

+ Mã hóa node q

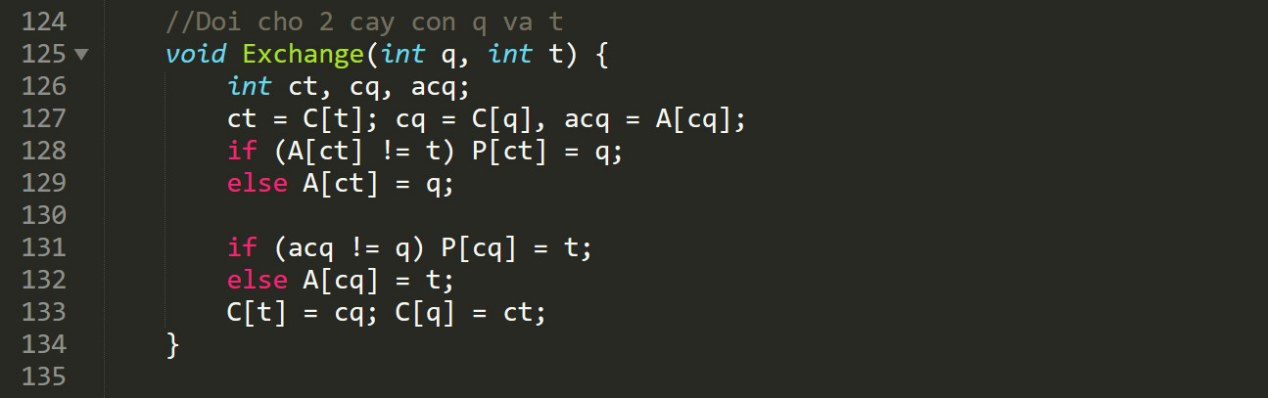
Gán đường đi từ node đến node gốc





- Hàm Exchange

Đảo vị trí 2 cây con có cùng trọng số



***- Hàm Decode:*** thực hiện giải mã chuỗi nhị phân thành chuỗi kí tự

Nếu q là internal thì thực hiện vòng lặp

Đi đến node lá

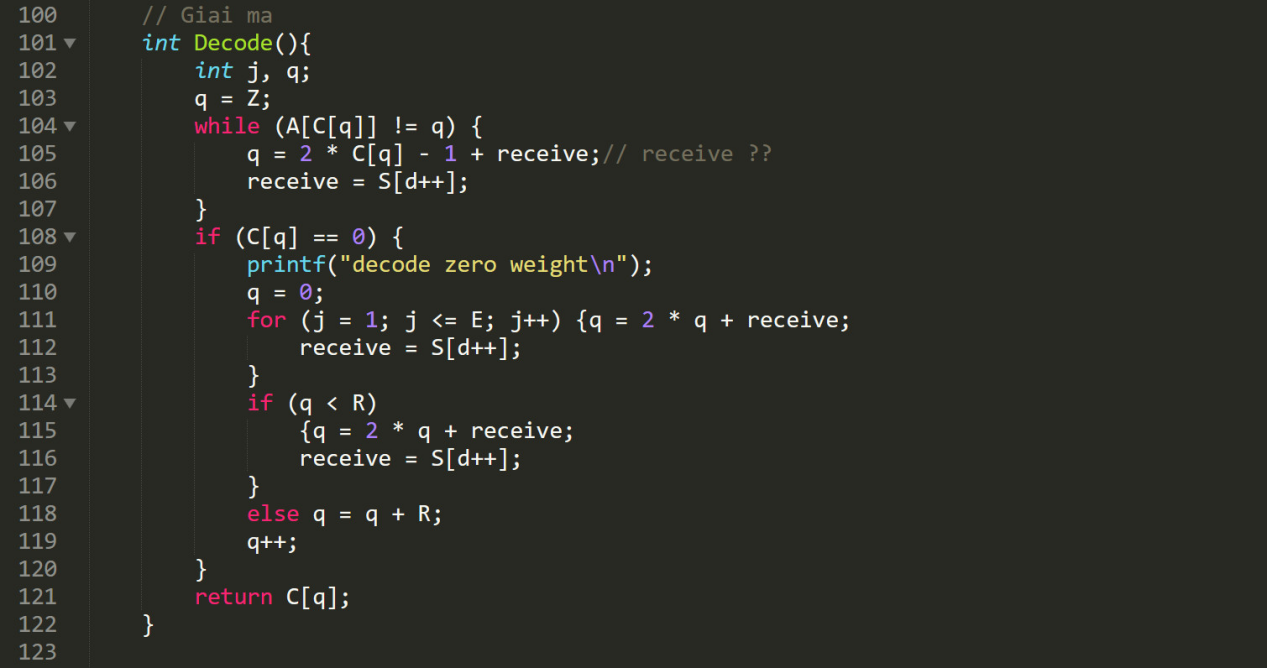
Nếu node lá là node biểu diễn cho node có trọng số 0 thì:

+ In ra “decode zero weight”

+ Thực hiện E bước đi có sẵn đi lên

+ Nếu q < R thì thực hiện đi lên 1 bước, Nếu không thì q = q + R

Trả về C[q];



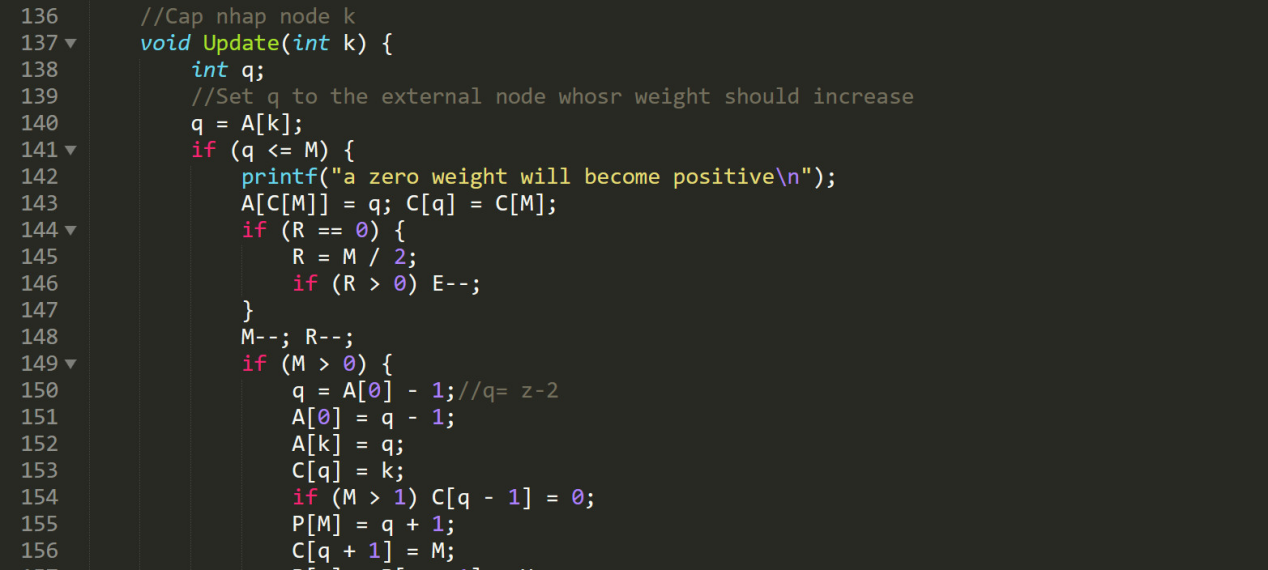
***Hàm Update:*** update ký tự ak vào cây

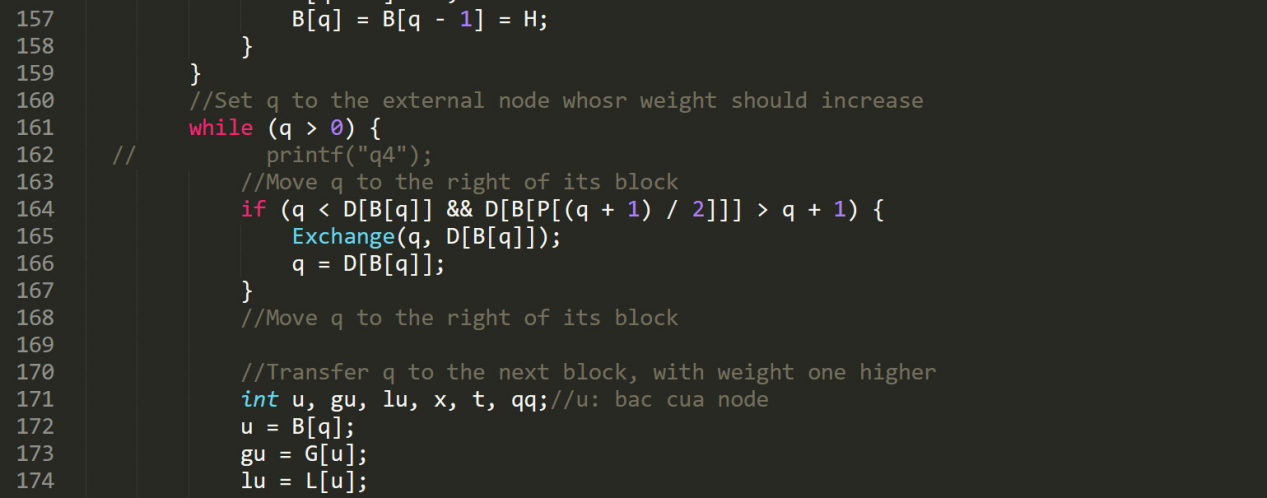
- Đặt q là node lá biểu diễn cho ký tự ak

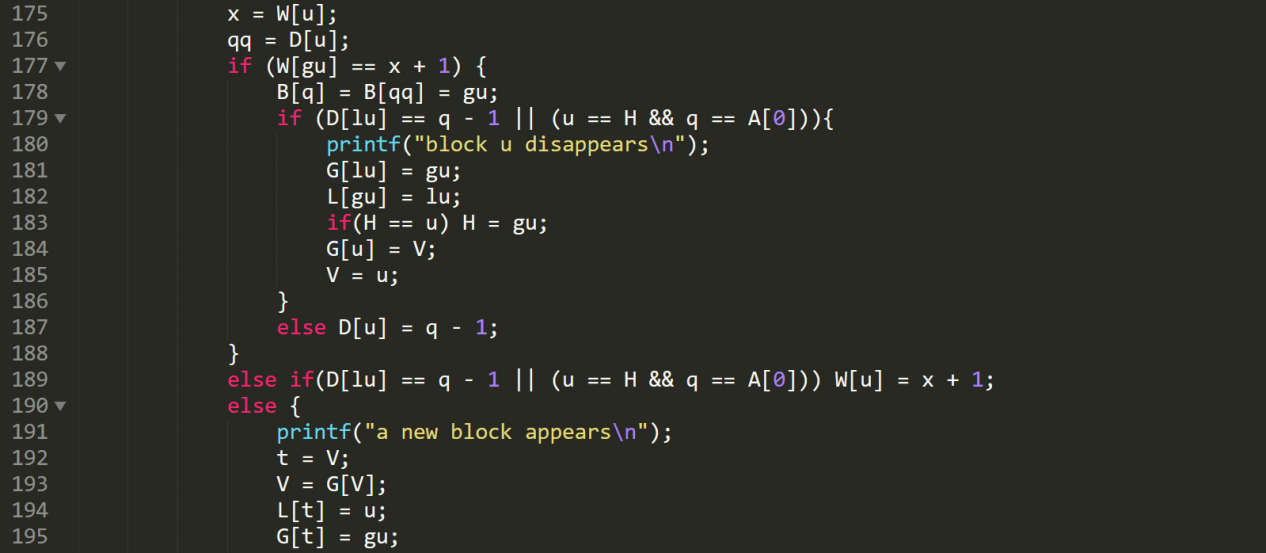
- Vòng lặp q > 0; với bước nhảy p = P[(q+1) /2];

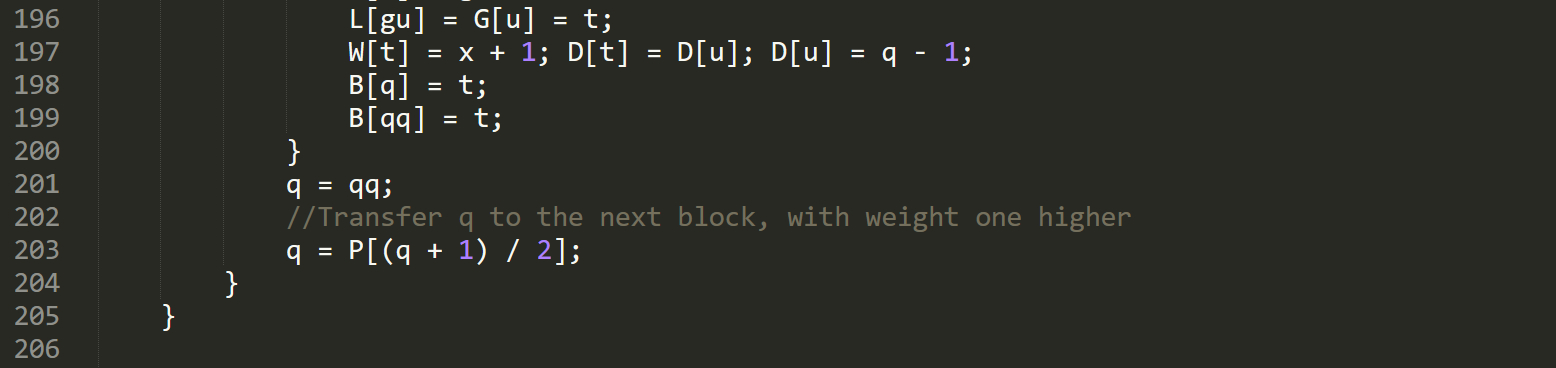
+ Di chuyển node q sang bên phải

+ Di chuyển node sang block tiếp theo có trọng số lớn hơn









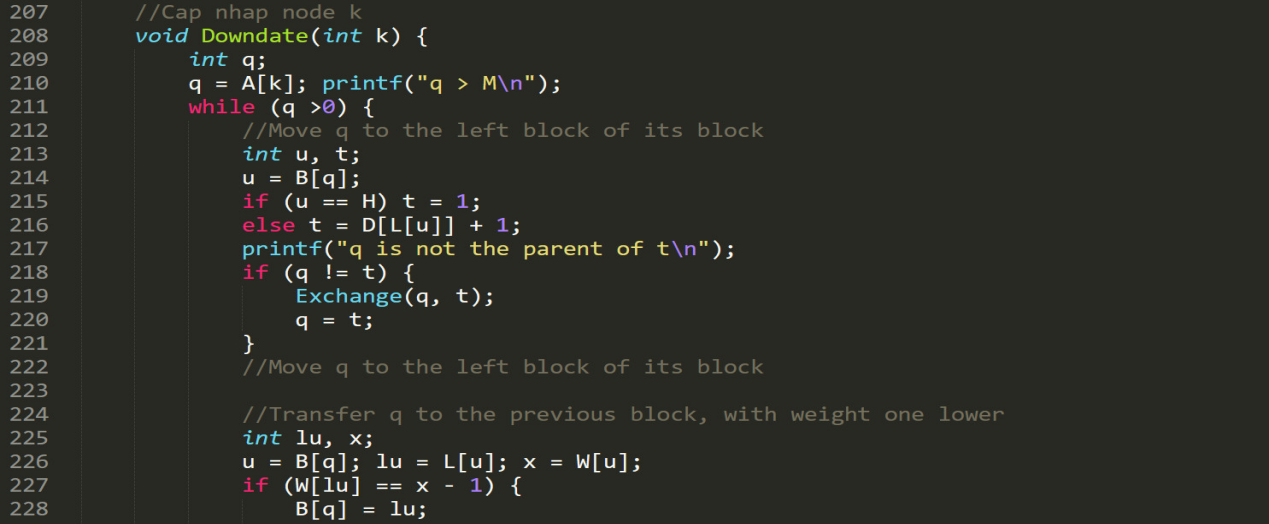
***- Hàm Downdate***

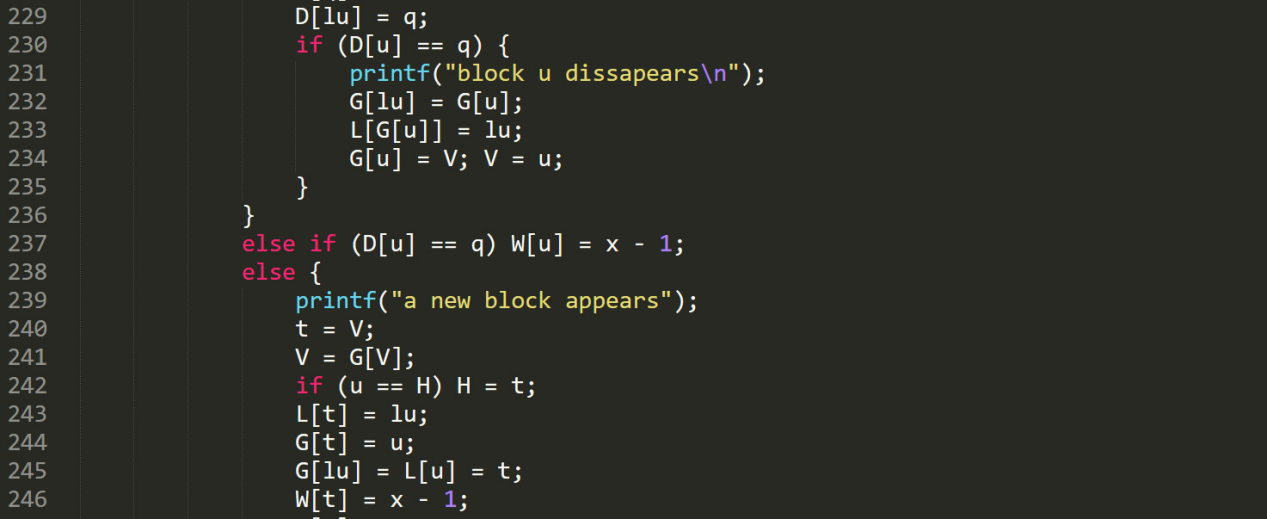
Đặt q là node lá biểu diễn cho kí tự ak

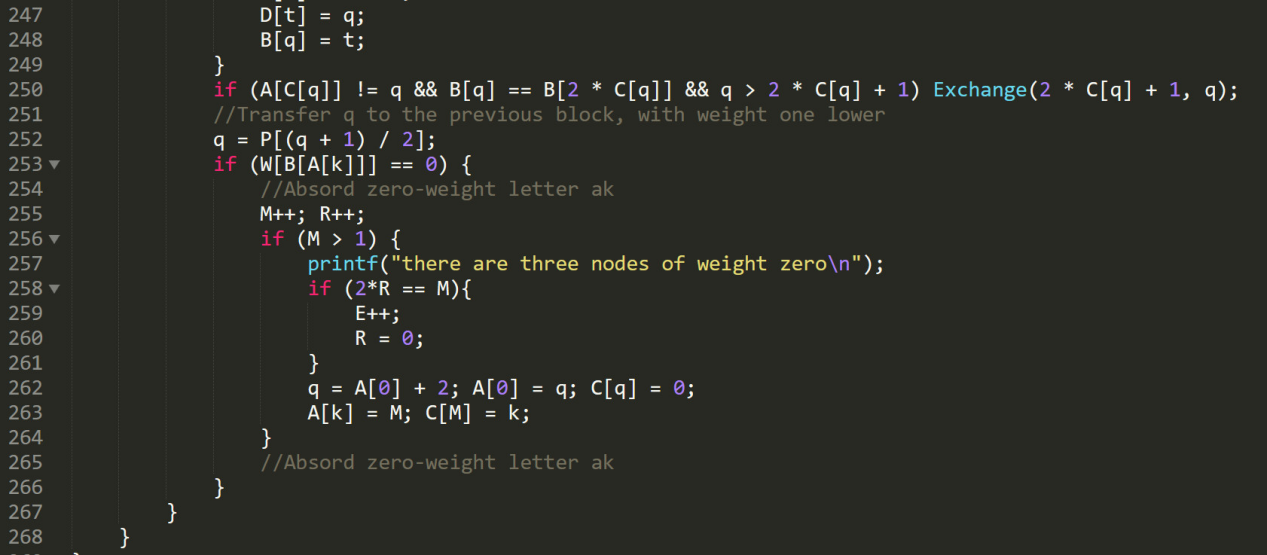
Vòng lặp q > 0, bước nhảy q = P[(q+1) / 2]

+ Di chuyển q sang bến trái khối của nó

+ Di chuyển số sang block tiếp theo có trọng số nhỏ hơn

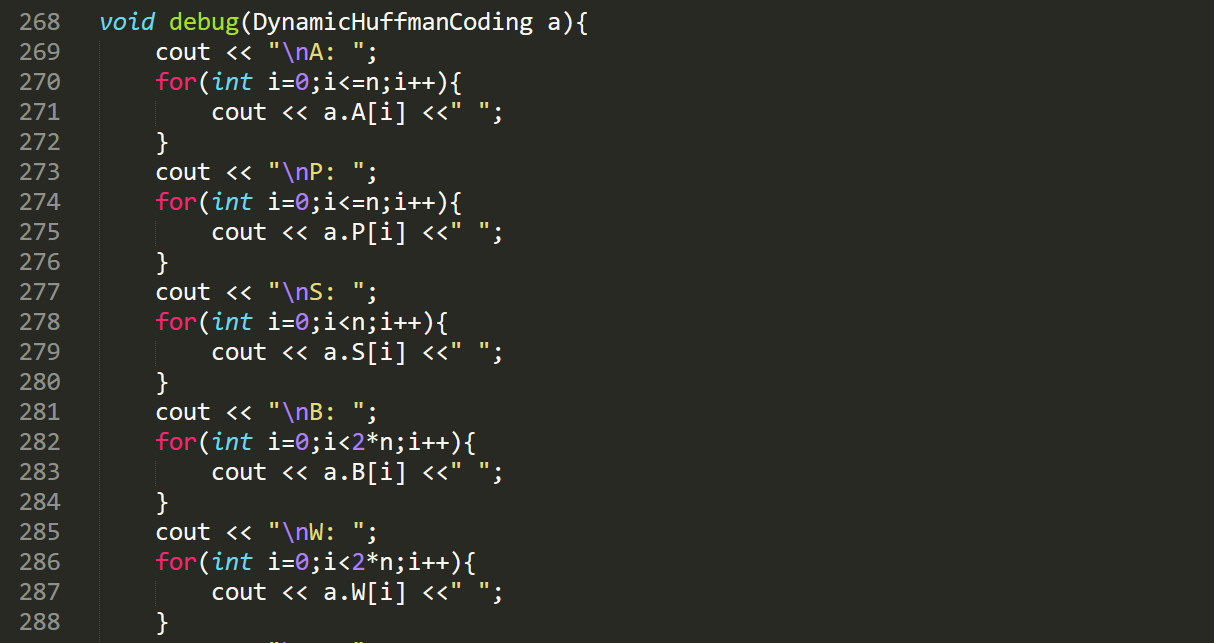


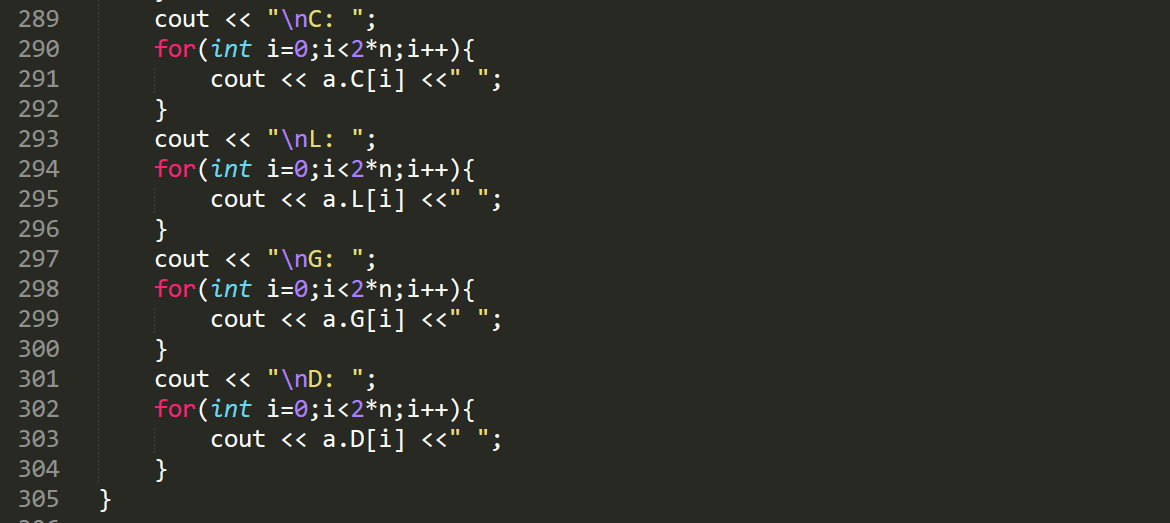




***- Hàm Debug***

Kiểm tra xem các biến có chạy đúng theo các bước thực hiện hay không



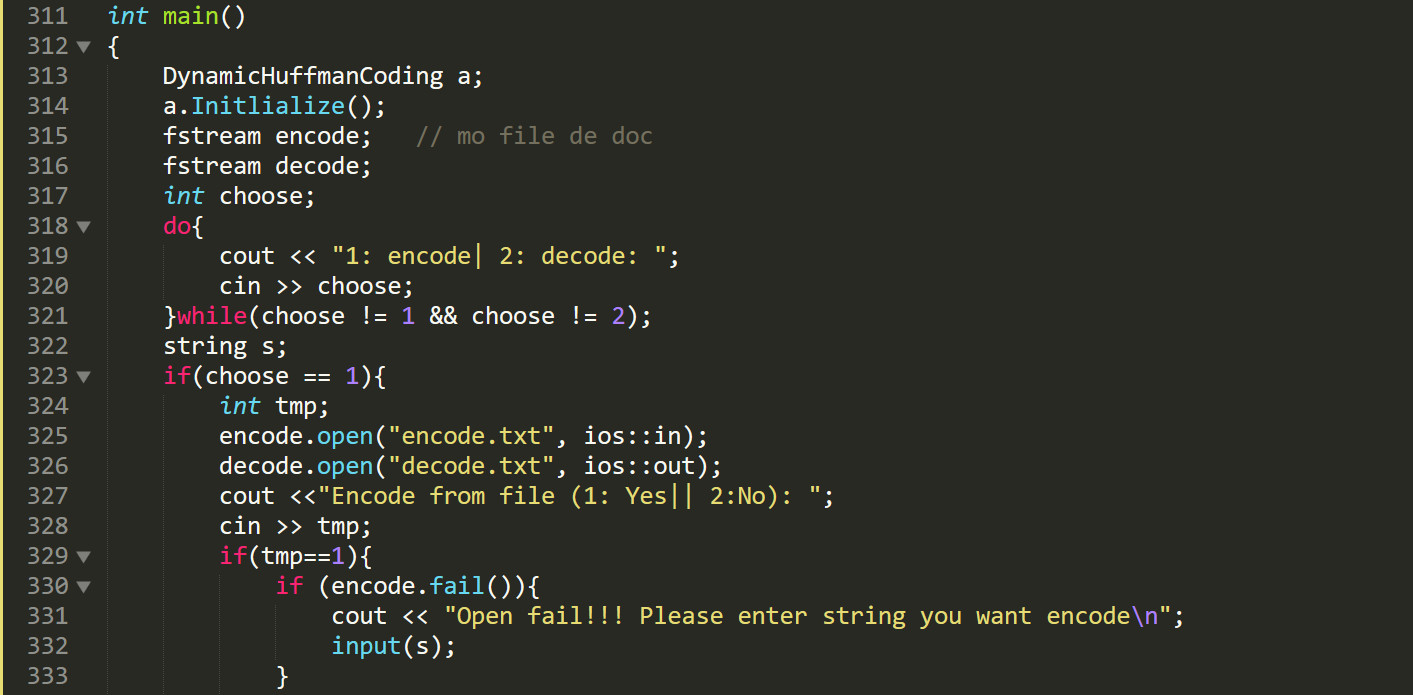


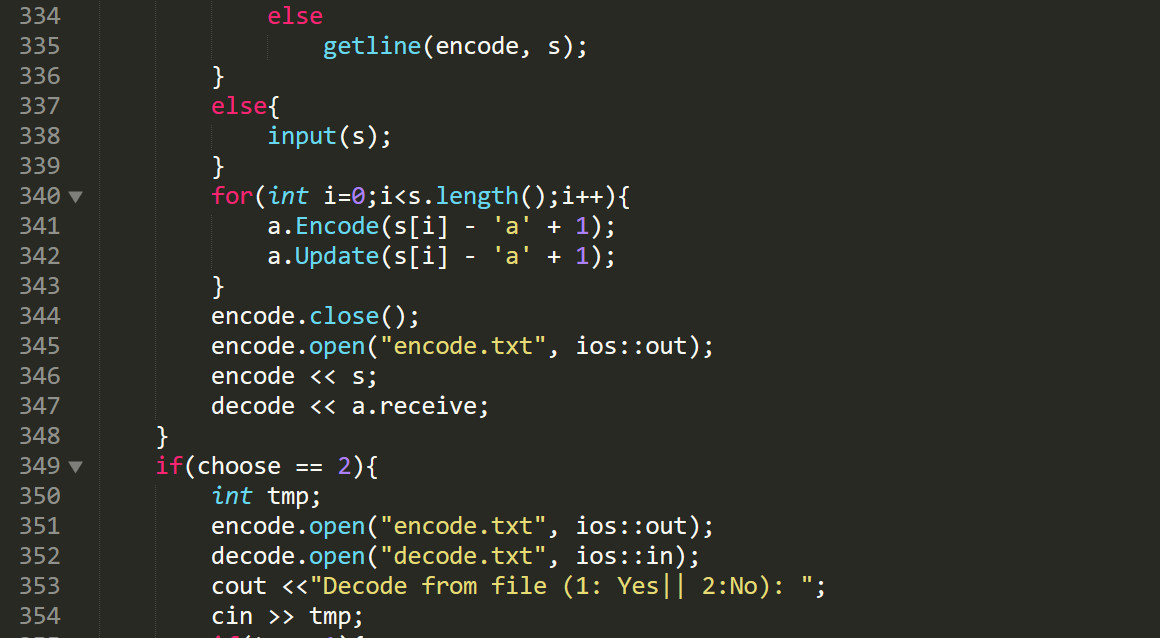
***- Hàm main***

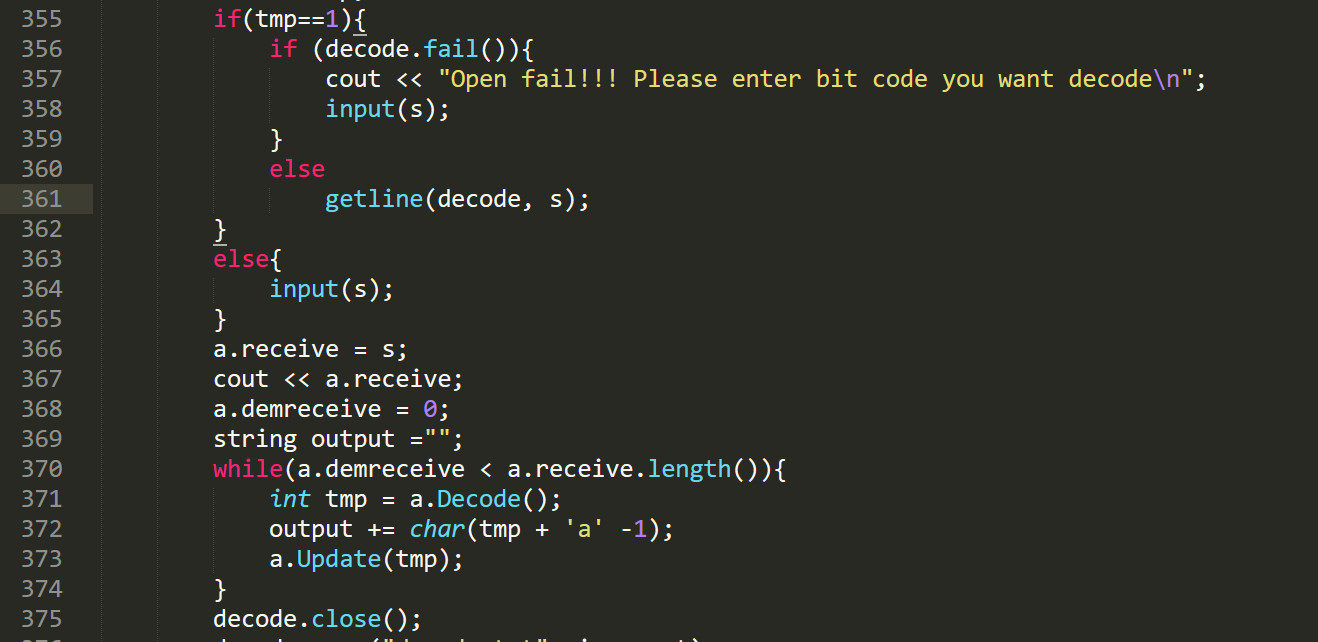
Thực hiện lệnh Encode và Decode theo yêu cầu:

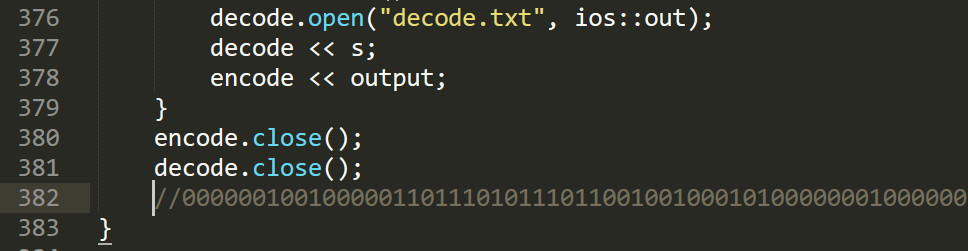
+ Chọn 1: để thực hiện mã hóa nội dung được nhập vào hoặc nội dung đã có sẵn trong file encode.txt. Chuỗi nhị phân sẽ được xuất ra file decode.txt

+ Chọn 2: để thực hiện giải mã chuỗi đã được mã hóa trong file decode.txt, nội dung sẽ được xuất ra file encode.txt









**2. Các ứng dụng của thuật toán**

- Nén hình ảnh mà không làm mất đi dữ liệu vốn có: Dynamic Huffman Coding được dùng trong các hình ảnh về y tế, hình ảnh về vệ tinh cần độ phân giải cao để xem rõ chính xác

- Nén văn bản: từ 1 chuỗi văn bản cho trước, mã hóa thành 1 chuỗi nhị phân của các ký tự, được dùng trong các định dạng nén thông thường như Gzip, vv…

- Nén âm thanh là một lĩnh vực ứng dụng khác được hưởng lợi rất nhiều từ mã hóa Huffman vì không mất dữ liệu. Các mẫu âm thanh thường được ghi dưới dạng 16 bit/channel. Có một số kỹ thuật nâng cao như đệ quy và biểu diễn nhiều ký hiệu với xác suất đơn vẫn có thể hoạt động bằng cách mã hóa các số này như thế này. Nhưng, đây không phải là những kỹ thuật rất phổ biến.

- Truyền văn bản và fax

- Dùng trong mã hóa thống kê

**3. Hướng phát triển của thuật toán**

**-** Cắt giảm, gộp biến và hàm để tối uu hóa code, giảm thiểu vòng lặp để tăng tốc độ chương trình

**-** Mở rộng thuật toán, cho phép mã hóa chuỗi có nhiều kí tự hơn, giảm thời gian chạy chương trình một cách tối đa

- Dynamic Huffman Coding thường được sử dụng để nén dữ liệu, vì thế cần phải phát triển thêm để nó xử lí được các loại ảnh với độ phân giải ngày càng cao và âm thanh với tần số đặc biệt để con người có thể nghiên cứu được thế giới xung quanh