TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

──────── \* ───────

**BÀI TẬP LỚN**

MÔN: LẬP TRÌNH MẠNG

**Đề tài: Mã hóa với thuật toán Huffman động**

Giảng viên hướng dẫn : **Bùi Trọng Tùng**

Sinh viên thực hiện : **Hoàng Ngọc Sinh**

Lớp : **AS1 Việt Nhật – K52**

***Hà Nội, tháng 11 năm 2010***

Muc Luc

[*Lời nói đầu* 3](#_Toc278446065)

[I.CƠ SỞ LÝ THUYẾT: 4](#_Toc278446066)

[**1.****TCP concurrent single server:** 4](#_Toc278446068)

[**2.****Thuật toán Huffman động:** 6](#_Toc278446069)

[**3.****Cây huffman động:** 9](#_Toc278446071)

[**4.****Thuật toán mã hóa và giải mã:** 11](#_Toc278446072)

[II.CÀI ĐẶT, KIỂM THỬ: 12](#_Toc278446073)

[**1.****Cài đặt:** 12](#_Toc278446074)

[**2.****Kiểm thử:** 12](#_Toc278446075)

[III.KẾT LUẬN: 13](#_Toc278446076)

[**1.****Đánh giá:** 13](#_Toc278446077)

[**2.****Hướng phát triển:** 13](#_Toc278446078)

I[V.TÀI LIỆU THAM KHẢO: 13](#_Toc278446079)

**Lời nói đầu**

Cùng với sự phát triển của công nghệ thông tin, mạng máy tính đã và đang dần đón vai trò quan trọng trong cuộc sống hiện nay. Đi đôi với sự bùng nổ toàn cầu về công nghệ thông tin, lượng dữ liệu ngày càng khổng lồ khiến cho việc quản lý dữ liệu đơn lẻ trở nên phức tạp hơn. Mạng máy tính ra đời đã giúp chúng ta đơn giản hóa việc quản lý, dễ dàng chia sẽ dữ liệu cũng như tài nguyên phần cứng. Để đảm bảo cho hệ thống mạng hoạt động hiệu quả ổn định, bên cạnh thiết bị mạng cũng như cấu hình mạng chúng ta cần phải lập trình cho sự hoạt động đó. Điều này khẳng định vai trò quan trọng của bộ môn lập trình mạng.

Bên cạnh sự phát triển của mạng máy tính, thì việc chia sẽ dữ liệu cũng trở nên vô cùng quan trọng. Để dễ dàng thuận tiện và nâng cao sự bảo mật cho việc chia sẽ dữ liệu, các thuật toán mã hóa được ra đời. Trong đó thuật toán Huffman động là một trong những thuật toán kinh điển và được ứng dụng trong rất nhiều bài toán hiện nay. Đề tài này tập trung nghiên cứu và cài đặt thuật toán mã hóa Huffman đông, tiến hành kiểm thử qua mô hình client/server .

Báo cáo được chia thành 3 phần lớn.

Phần I: Cơ sở lý thuyết

1. Concurrent single server
2. Thuật toán Huffman động

Phần II: Xây dựng chương trình

Phần III: Cài đặt và kiểm thử

PhầnIV: Kết luận và hướng phát triển

Từ sự giúp đỡ tận tình của thầy Bùi Trọng Tùng về mặt lý thuyết em đã xây dựng chương trình mã hóa sử dụng thuật toán Huffman động thông qua giao tiếp giữa client và server. Trong đề tài em đã sử dụng Concurrent single server . Đây là báo cáo ngắn gọn của em về đề tài của mình

1. Cơ sở lý thuyết
2. Concurrent single server
   1. Socket

* Socket là một phương pháp kết nối truyền thông giữa một chương trình yêu cầu dịch vụ (Client) và chương trình cung cấp dịch vụ(Server).
* Quá trình trao đổi dữ liệu thông qua socket
  + Server tạo ra socket gắn với cổng của nó và chờ kết nối từ client
  + Client cũng tạo ra 1 socket để kết nối với socket do server tạo ra.
  + Sauk hi kết nối client và server trao đổi dữ liệu với nhau bằng cách đọc vào hoặc ghi từ socket của mình
  1. TCP
* TCP cung cấp kênh truyền thông giữa client và server
* TCP đảm bảo tính tin cậy: khi TCP gửi dữ liệu đến cho TCP khác nó yêu cầu xác nhận (ACK).
* TCP cung cấp khả năng điều khiển luồng
* TCP hoạt động theo kiểu song công
  1. Stream socket APIs

Hình 2. Sơ đồ hoạt động của client server

* socket()
  + Tạo socket dựa trên Domain
  + Trả về một socket ID
* bind()
  + Gán định danh cho socket
  + Kết nối socket với 1 địa chỉ IP và cổng tương ứng
* listen()
  + Lắng nghe kết nối từ phía client
* accept()
  + Chấp nhận một kết nối từ client.
  + Có 3 cách để chấp nhận kết nối từ client
    - Iterating server
    - Select server
    - Forking server
* connect()
  + Client gửi yêu cầu kết nối tới server
* send()/receive()
  + Truyền nhận dữ liệu
* close()
  + Đóng kết nối
* Một số APIs hỗ trợ khác
  + gethostname :Trả về tên hệ thống
  + gethostbyname:ánh xạ giữa hostname và IP
  + htons, htonl, ntohs, ntohl:Thao tác dữ liệu theo byte
  + inet\_ntoa(), inet\_aton(), inet\_addr():Biến đổi địa chỉ IP từ chuỗi thành struct add\_r và ngược lại
* Cấu trúc địa chỉ socket

Với IP4

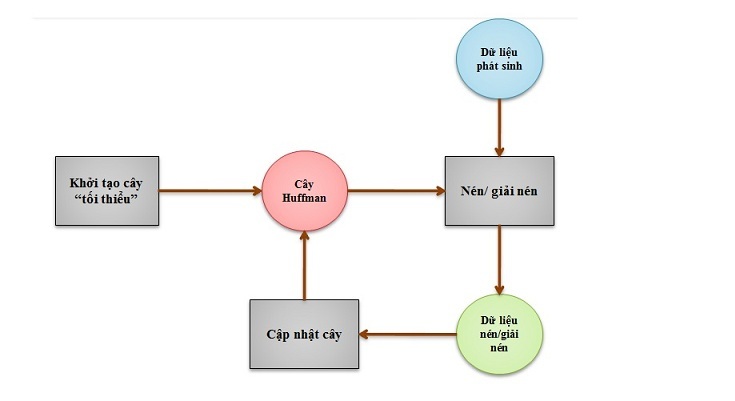
|  |
| --- |
| struct in\_addr{  in\_addr\_t s\_addr;  };  struct sockaddr\_in{  unit\_8t sin\_len;  sa\_family\_t family\_sin;  in\_port\_t sin\_port;  struct in\_addr sin\_addr;  char sin\_zero[8];  }; |

1. Thuật toán Huffman động
   1. Sự hạn chế của Huffman tĩnh

* Trong quá trình mã hóa file cần 2 lần duyệt file -> chi phí mã hóa cao
* Cần phải lưu trữ thông tin để giải mã nên sẽ làm tăng kích thước file mã hóa
* Dữ liệu mã hóa phải có sẵn -> không mã hóa được dữ liệu phát sinh theo thời gian thực
  1. Ý tưởng

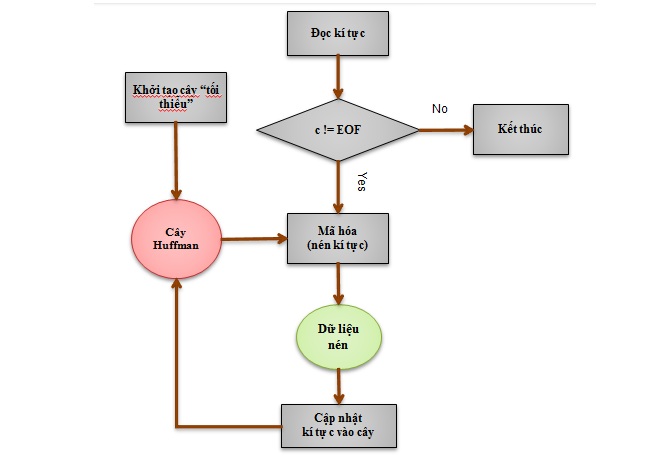
Để khác phục được những nhược điểm chính của thuật toán Huffman tĩnh người ta đưa ra thuật toán Huffman động(Adaptive Huffman).

* Sử dụng một vài bit để biểu diễn một ký tự
* Độ dài mã bit không giống nhau
  + Ký tự xuất hiện ít: Biểu diễn bằng mã dài
  + Ký tự xuật hiện nhiều: Biểu diễn bằng mã ngắn
* Tạo sẵn một cây “tối thiểu” ban đầu. Dữ liệu mã hóa sẽ dần được cập nhật vào cây.
  1. Ưu điểm của thuật toán Huffman động
* Không cần tính trước số lần xuất hiện của ký tự
* Quá trình nén chỉ cần một lần duyệt file
* Không cần lưu thông tin cho quá trình giải nén
* Nén “online” dữ liệu phát sinh theo thời gian thực.
  1. Thuật toán tổng quát
* Khởi tạo cây tối thiểu ban đầu, cây sẽ được cập nhật dần dựa trên dữ liệu phát sinh trong quá trình mã hóa hoặc giải mã

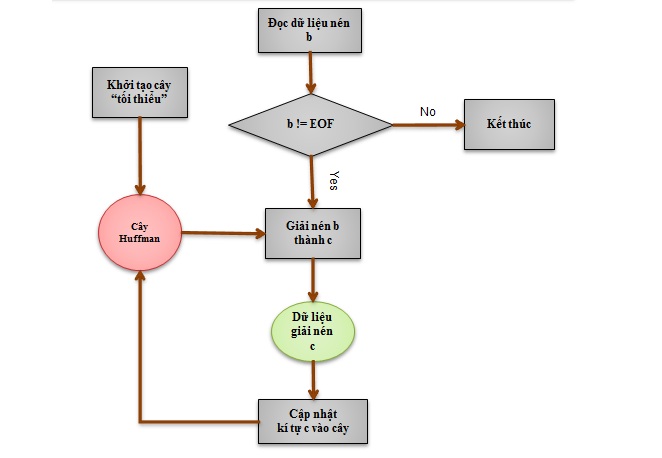


Hình 3.Sự phối hợp giữa việc dùng cây (cho thuật toán nén/giải nén) và cập nhật cây

* Thuật toán mã hóa



* Thuật toán giải mã

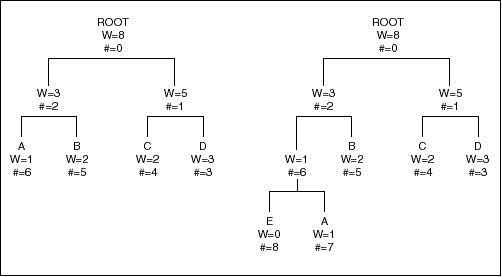


1. Cây Huffman động
   1. Giới thiệu

Một cây nhị phân n nút lá được gọi là cây Huffman động nếu nó thỏa mãn các tính chất sau

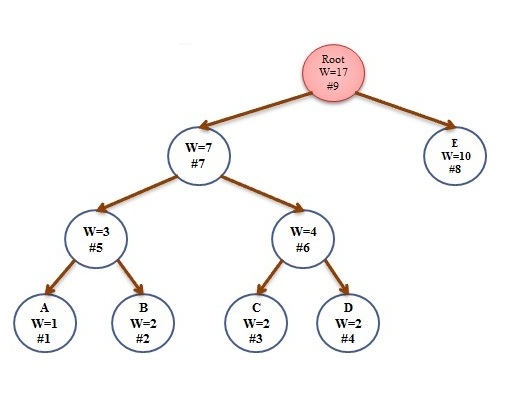
* Các node lá có trọng số wi ≥ 0, i ∈ [1,n]
* Các node nhánh có trọng số bằng tổng trọng số của các node con của nó

Ví dụ:



Hình 5.Cây Huffman động

* 1. Tính chất sibling
* Mỗi node ngoại trừ node gốc đều tồn tại node anh/em(có cùng node cha)
* Khi sắp xếp các node theo thứ tự tăng dần trọng số thì các node luôn kề bên node anh/em của nó.
* Ví dụ



* Trong ví dụ trên khi tap sắp xếp các nút theo chiều tăng dần của trọng số ta được

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Thứ tự | #1 | #2 | #3 | #4 | #5 | #6 | #7 | #8 | #9 |
| Wi | 1 | 2 | 2 | 2 | 3 | 6 | 7 | 10 | 17 |
| Giá trị | A | B | C | D |  |  |  | E | Root |

* 1. Cách hình thành và cập nhật cây
* Bước 1: Khởi tạo cây tối thiểu chỉ có node Escape(node 0)
* Bước 2:cấp nhật từng ký tự vào cây theo quy tắc
  + Nếu ký tự chưa có trong cây thêm với node lá
  + Nếu ký tự có rồi tăng trọng số node lên 1
  + Cập nhật trọng số của các node liên quan trên cây
* Thuật toán cập nhật trọng số
  + Tăng trọng số của node lá lên 1
  + Đi từ node lá đến node gốc tăng trọng số của các node lên 1. Kiểm tra tính chất anh em và hiệu đỉnh lại nếu có vi phạm.
  1. Cách giải quyết vi phạm

Khi ta tăng trọng số của cây có 2 vấn đề có thể xảy ra. Thứ nhất là tràn số thứ 2 là vi phạm tính chất anh em của cây. Vì vậy chúng ta phải hiệu chỉnh lại để đảm bảo cây vẫn là cây Huffman động.

* Đối với vi phạm tính chất an hem
  + Xác định node vi phạm
    - Gọi x là node hiện hành
    - So sánh x với các node tiếp theo sau theo thứ tự từ trái sang phải từ dưới lên trên
    - Nếu tồn tại node y trong đó trọng số của y nhỏ hơn trọng số của x thì x là node vi phạm
  + Hiệu chỉnh cây Huffman động
    - Node x là node vi phạm
    - Tìm node y xa nhất và có trọng số cao nhất thỏa mãn trọng số của nó vẫn nhỏ hơn x.
    - Hoán đổi vị trí của 2 node x và y cho nhau
    - Cập nhật lại trọng số các node cha tương ứng
    - Lặp lại các bước trên cho đến khi không còn node vi phạm
* Với vấn đề tràn số
  + Vấn đề này có thể xảy ra khi chúng ta cập nhật trọng số cho cây.
  + Thuật toán xử lý tràn sô
    - Khi cập nhật trọng số kiểm tra trọng số node gốc
    - Nếu trọng số node gốc >MAX\_VALUE
      * Giảm trọng số node gốc(chia cho 2)
      * Cập nhật lại trọng số các node nhánh
      * Kiểm tra tính chất anh em và hiệu chỉnh lại cây

1. Thuật toán mã hóa và giải mã
   1. Mã hóa

|  |
| --- |
| initialize\_model();  while(c != EOF)  {  c = getchar(inputfile);  encode(c, outputfile);  update\_model(c);  } |

* Mã hóa một ký tự

**encode (c, outputfile)**

* + - * Nếu c chưa có trong cây
      * Duyệt cây T tìm mã bit của Escape ghi lên outputfile.
      * Ghi tiếp 8 bit mã ASCII của c lên outputfile.
      * Thêm node c vào cây và cập nhật lại cây.
      * Nếu c đã có trong cây:
      * Duyệt cây T tìm mã bit của c và ghi lên outputfile
      * Tăng trọng số của node c lên 1 và cập nhật lại cây
  1. Thuật toán giải nén

|  |
| --- |
| inputfile: dữ liệu ở dạng nén  outputfile: dữ liệu giải nén  initialize\_model();  while((c = decode(inputfile)) != EOF)  {  putchar(c, outputfile);  update\_model(c);  } |

* + - Thuật toán giải mã kí tự c:
      * **decode(inputfile)**
      * **b = getchar(inputfile);**
      * Bắt đầu từ vị trí hiện tại trên inputfile.
      * Lấy từng bit của b, duyệt trên cây (b==0: trái, b==1: phải).
      * Nếu đi đến node lá x, return x.char
      * Nếu đi đến node Escape:
      * c = lấy 8 bit tiếp theo từ inputfile
      * Thêm c vào cây, cập nhật lại cây.
      * return c

1. Cài đặt kiểm thử
2. Cài đặt
   1. Một số hàm quan trọng

//Hàm mã hóa

void Endcode(char \*source, char \*output);

//Hàm giải mã

void Decode(char \*source, char \*output);

//Hàm updateTree theo thuật toán FGK

void update\_treeFGK( int c );

1. Kiểm thử

Các chức năng như đăng nhập, đăng ký hoạt động tốt ổn định.Thuật toán nén rất tốt đối với file text(nén 54.54%). Tuy nhiên đối với những file đã được né hoặc không phải file text thì kết quả không được tốt(thường tăng dung lượng).

1. Kết luận
   1. Ưu điểm
      * Chương trình hoạt động ổn định chính xác
      * Tốc độ nén nhanh, dung lượng nén cũng đạt yêu cầu(54.54% với file text)
      * Nén được mọi loại file và không làm mất cấu trúc của nó khi giải nén
      * Nhìn chung đạt các yêu cầu chức năng mà chương trình phải đáp ứng

2. Tồn tại

* Chưa cải tiến được thuật toán huffman động
* Chưa tối ưu hóa quá trình nhận và gửi file giữa client và server.
  1. Hướng phát triển
     + Xây dụng chương trình gửi file theo P2P(nhằm tăng tốc độ truyền gửi file)
     + Cải tiến thuật toán Huffman để có hiệu quả cao hơn

1. Tài liệi ham khảo

***Mark Nelson, Jean-Louo Gailly***

The data compression book

***Dan Chen Yi-Jen Chiang Nasir Memon Xiaolin Wu***

Alphabet Partitioning Techniques for emi-Adaptive Huffman Coding of Large Alphabets.

Linux socket programming