**BIẾN ĐỔI CHUỖI THÀNH MỘT CHUỖI CÓ KẾT QUẢ CỐ ĐỊNH**

**Trần Quốc Việt1, Phạm Thanh Hòa1**

*1Đại học Sư Phạm Kỹ Thuật, TP. Hồ Chí Minh*

Tóm tắt:

Hiện nay, trong thời điểm cuộc cách mạng công nghiệp lần thứ 4 đang được diễn ra và tâm điểm của nó chính là ứng dụng công nghệ vào đời sống, đồng nghĩa với việc đó là thông tin sẽ ở khắp mọi nơi. Đề tài chúng em chọn là một trong những nền tảng của bảo mật thông tin, **biến đổi chuỗi thông tin**, nên vì vậy cùng với xu hướng đó nhóm chúng em quyến đinh chọn đề tài này trở thành đồ án cuối kỳ của nhóm. Dựa trên phương pháp **phân tích, giả thuyết** và **thực nghiệm**, dựa vào phân tích xác định được đầu vào và đầu ra của bài toán, sau đó mở rộng đầu ra bằng cách đặt ra các giả thuyết có thể được từ đầu vào, và cuối cùng là đưa vào thực nghiệm từ phương án đã đưa ra. Sau thực nghiệm chúng em có thể đưa ra kết quả việc ứng dụng trí tuệ nhân tạo vào bài toán có thể giải được bài toán **nhanh hơn** …Việc áp dụng trí tuệ nhân tạo vào việc giải các bài toán biến đổi, hay mở rộng ra có thể là các bài toán mã hóa, hoặc giải mã có thể làm giảm thời gian chờ cho việc này.

Từ khóa: Biến đổi chuỗi thông tin; phân tích; giả thích; thực nghiêm; nhanh hơn.

1. **ĐẶT VẤN ĐỀ.**

Chương trình biến đổi một chuỗi ký tự sẽ là chương trình nhằm giải quyết các vấn đề về chuỗi, vấn đề này xoanh quanh việc biến đổi chuỗi ký tự. Chuỗi ký tự này sẽ được nhập từ bàn phím các ký tự có thể giải được sẽ nằm trong tổ hợp ký tự: A, B, C, E. Kết quả biến đổi sau cùng sẽ là chuỗi “E”, và nếu chuỗi không thể biến đổi được thì sẽ có thông báo ngươc lại người dùng. Vì hướng tới điều cơ bản chương trình sẽ xử dụng ngôn ngữ C/C++ để lập trình, viết trên giao diện console và được viết theo hướng thủ tục.

1. **PHƯƠNG PHÁP**
   1. Dữ liệu đầu vào.

Chương trình sẽ có đầu vào là một chuỗi ký tự có giới hạn là 100 ký tự, nhóm ký tự này sẽ được nhập từ bàn phím máy tính. Các ký tự chương trình sẽ hiểu là A, B, C, E. Nếu có bất kì ký tự nào khác nhập vào chương trình sẽ đưa ra kết quả không thể giải.

* 1. Phương pháp xử lý.

Theo bài toán đặt ra, chuỗi dữ liệu sẽ biến đổi nhờ bốn quy luật sau: AB =>BC, AC=>E, BB=>E, Ex=>x (trong đó x là một ký tự bất kỳ). Vì lý do đó chương trình nếu giải quyết theo thuật toán breath-first search sẽ có thể cho một cái nhìn trực quan tốt hơn các thuật toán tìm kiếm mù khác, lý do cho việc trên chính là mỗi lần biến đổi chuỗi sẽ giảm độ dài dần lúc này nếu tiếp tục tìm kiếm trên chuỗi đã biến đổi có thể tạo ra hàng loạt điểm quy hồi có thể làm rối chương trình khó để chỉnh sửa nếu tính logic của chương trình bị sai. Sau đó tìm ra các vấn đề làm chậm chương trình, giải quyết chúng để chương trình có thể giải bài toán nhanh hơn.

* 1. Mô tả chi tiết thuật toán.

Mô tả chương trình hoạt động:

* Giải thích chung: Chương trình sẽ sử dụng 2 list được gọi là open và close, mỗi khi phát hiện một state (vị trí có thể áp dụng luật), state đó sẽ được đưa vào open và ngồi chờ cho tới khi được lấy ra và ghép vào chuỗi kết quả.

Bắt đầu

Xuất kết quả

Thuật toán

Current->state   
==   
goal->state

Current = FIFO\_pop(open)  
thêm current vào close

Không có kết quả

Thêm root vào open

Nếu số lượng node của open >0

Khởi tạo Node list open, close

Hình 1: Lưu đồ hàm giải tổng quát

* String (BFS).cpp: Chương trình này được áp dụng thuật toán Breath-first Search để tìm kiếm kết quả. Chương trình sẽ áp dụng từng luật một vào mỗi state để đưa ra kết quả tiếp theo.

I = 0

Sai

I<=strlen(current->state)-1

Đúng

Sai

Nếu current->state[i] có thể áp dụng các luật

Đúng

Tạo node con đã áp dụng các luật từ current

Sai

Node con không tồn tại trong open lẫn close

Đúng

Thêm node con vào cuối list open

I++

Hình 2: Lưu đồ thuật toán BFS

* String (Astar).cpp: Trong đoạn chương trình này gồm 2 thuật toán A\* khác nhau. Đối với A\* đầu tiên chương trình sẽ ưu tiên rút gọn, làm ngắn chuỗi đi (chọn các luật có thể biến đổi trực tiếp thành E hoặc gián tiếp, sau đó rút gọn E làm chuỗi ngắn lại), sau đó sử dụng lại thuật toán BFS xử lý chuỗi còn lại, sau đó lại lập lại việc rút gọn chuỗi và giải. Còn về A\* thứ hai, chương trình sẽ tính điểm G, điểm g được tính theo công thức: độ dài chuỗi + path\_cost - số lượng các luật có thể làm ngắn chuỗi (AC, BB, ABAB, E).

I = 0

BFS

I<=strlen(current->state)-1

Sai

Đúng

Sai

Current->state[i] có thể áp dụng các luật  
(ABAB,BB,AC,E)

Đúng

Tạo node con đã áp dụng luật

Sai

I++

Node con không tồn tại trong open lẫn close

Đúng

Thêm node con vào đầu list open

Hình 3: Lưu đồ thuật toán A\* thứ nhất

I = 0

I<=strlen(current->state)-1

Sai

Đúng

Sai

Nếu current->state[i] có thể áp dụng các luật

Đúng

Tạo node con đã áp dụng các luật từ current

Tính điểm heuristic của node con

Sai

Node con không tồn tại trong open lẫn close

Đúng

Thêm node con vào list open(theo thứ tự điểm heuristic tăng dần)

I++

Hình 4: Lưu đồ thuật toán A\* thứ 2

1. **KẾT QUẢ**

Kết quả mẫu thử (thời gian chạy)

|  |  |  |  |
| --- | --- | --- | --- |
| Mẫu thử | Breath-first Search | A\* (1) | A\* (2) |
| ABAB | 0.002s | 0.001-0.002s | 0.002s |
| EEEE | 0.002s | 0.001s | 0.001s |
| AAABAABC | 0.003 | 0.004s | 0.004s |
| ABAB (5) | >10s | 0.007-0.01s | 0.021-0.023s |
| ABAB (10) | >10s | 0.029-0.03s | 0.045s |
| ABC (15) | >10s | >10s (120s) | 0,001s |

Bảng 1: Bảng kết qua kiểm thử

Kết quả mẫu thử cho thấy thuật toán A\*(1) có kết quả chạy tốt nhất đối với cái mẫu thử có kết quả nhưng lại cho ra kết quả thấp hơn so với A\*(2) với các mẫu thử không thể giải, nguyên nhân có thể là do A\*(1) được tối ưu cho việc tìm kiếm kết quả có đáp án, nên vì đó list open sinh ra với số lượng lớn để kiểm ra và A\*(1) phải kiểm tra toàn bộ số node này. Ngược lại, A\*(2) cho ra kết quả cân bằng hơn vì các node được thêm vào list phải được kiểm tra qua nhiều giai đoạn số lượng node trong open nhỏ nên các mẫu thử không có đáp án có thể giải nhanh hơn so với A\*(1).

1. **KẾT LUẬN**

Về mặt ưu điểm, hiện tại chương trình có thể hoạt động ổn định đúng với mục tiên đề ra, thông qua các phép thử cho thấy các phép toán có kích thước nhỏ (khoảng 40 ký tự) có thể cho ra kết quả ngay lập tức. Còn nhược điểm hiện tại của chương trình, như đã nói ở trên các phép thử nhỏ có thể đưa ra kết quả ngay nhưng đối với các mẫu thử lớn thì chương trình sẽ chậm lại 2n lần, đối với chuỗi không có lời giải, chương trình phải quét toàn bộ chuỗi để có thể đưa ra kết luận. Vì vậy, nếu gặp các phép thử có kích thước lớn thì người dùng không thể phân biệt được liệu chương trình có thể giải được hay không vì thời gian chờ quá lâu. Vì vậy, chúng em muốn đề xuất một vài phương pháp để có thể cải thiện được chương trình. Thứ nhất, hiện tại chương trình có thể vân hành một cách ổn định, mượt mà, nhưng về mặt giao diện vẫn chỉ ở mức console nên chưa thân thiện với người dùng quen với những thao tác window, nên có thể dưa lên giao diện window người dùng sẽ thao tác trực quan với chương trình. Thứ hai, với các mẫu thử lớn chương trình sẽ hoạt động khá lâu, vì thế cần phải có thêm thời gian để có thể tìm được cách giải các mẫu thử đó nhanh hơn, và cần phải giải quyết các mẫu thử sai, làm sao để đưa ra kết luận nhanh hơn.

**TÀI LIỆU THAM KHẢO**

1. Peter Norvig, Stuart J. Russell “Artificial Intelligence: A Modern Approach” – 3rd Edition, page.71-103

Thông tin tác giả chịu trách nhiệm bài viết

Họ tên: Trần Quốc Việt, Phạm Thanh Hòa.

Đơn vị: Khoa Đạo tạo chất lượng cao, đại học Sư Phạm Kỹ Thuật TP. Hồ Chí Minh