**BIẾN ĐỔI CHUỖI CHỮ PHỨC TẠP THÀNH CHUỖI CHỮ ĐƠN GIẢN**

**Bùi Việt Duy1, Đinh Tuấn An2**

*1 Trường đại học Sư phạm Kỹ thuật TP.HCM*

**TÓM TẮT**

Biến đổi chuỗi chữ là một trong những ứng dụng rất là quan trọng cuộc sống như từ các văn bản Word, Excel, đường dẫn,... giúp chúng ta thực hiện các tác vụ một cách nhanh chóng hơn. Thực tế, việc biến đổi chuỗi được sử dụng rộng rãi bởi vì người dùng họ luôn muốn mọi thứ được đơn giản hóa và thuận tiện cho vệc lưu trữ dữ liệu. Vì vậy, việc xây dựng một phương pháp tự động dựa trên hệ thống trí tuệ nhân tạo nhằm rút gọn chuỗi dữ liệu sẽ giúp người dùng rút ngắn được thời gian làm việc cũng như là hiệu quả trong công việc. Chương trình này ứng dụng các thuật toán tìm kiếm DFS, A\* để đưa ra kết quả biến đổi chữ phức tạp thành đơn giản. Chúng em xây dựng và kiểm chứng thuật toán trên tập dữ liệu gồm 100 bộ thử, đã giải được 13%.

Từ khóa: biến đổi chuỗi; rút gọn chuỗi; đổi chuỗi phức tạp thành đơn giản; chuẩn hóa chuỗi.

**1. ĐẶT VẤN ĐỀ**

Việc lưu trữ dữ liệu là công việc cần dùng đến một dung lượng bộ nhớ lớn. Mà dung lượng bộ nhớ thì là hữu hạn cho nên việc lưu trữ dữ liệu thì cần phải hợp lý nhất có thể để lưu trữ trong bộ nhớ. Ngoài ra việc gặp lỗi tràn bộ nhớ[1] là điều rất hay xảy ra có thể gây ra một ngoại lệ truy nhập bộ nhớ máy tính và chương trình bị kết thúc, hoặc khi người dùng có ý phá hoại, họ có thể lợi dụng lỗi này để phá vỡ an ninh hệ thống. Cho nên việc tối ưu dữ liệu lưu trữ là điều tất yếu. Vì vậy, việc rút gọn hay đơn giản hóa các chuỗi dữ liệu sẽ giúp cho việc lưu trữ dữ liệu được tối ưu và thuận tiện hơn. Trong những năm gần đây, trí tuệ nhân tạo (AI) được ứng dụng ngày càng nhiều trong mọi lĩnh vực để cải thiện trình độ chuyên môn và hiệu quả công việc lâm sàn. Do đó, việc ứng dụng AI để xây dựng chương trình biến đổi chuỗi dữ liệu tự động là một yêu cầu cấp thiết hiện nay.

Các phần còn lại của bài báo được tổ chức như sau. Phần 2 trình bày về phương pháp được sử dụng, bao gồm tập dữ liệu, các phương pháp xử lý, Depth first search (DFS), A\* . Phần 3 trình bày kết quả của quá trình biến đổi chuỗi áp dụng trên tập kiểm tra. Phần 4 tổng kết lại nghiên cứu này.

**2. PHƯƠNG PHÁP**

**2.1 Dữ liệu**

Sử dụng dữ liệu đầu vào cho chuỗi chữ ngẫu nhiên với độ dài và số lần thực hiện tùy chọn. Điều kiện đầu vào cho mỗi chuỗi chữ là nằm trong các kí tự ‘A’, ‘B’, ’C’, ‘E’. Bảng biểu 1 tổng kết số lượng chuỗi dữ liệu có thể rút gọn thành công với số lượng chuỗi và số lượng kí tự trong một chuỗi dữ liệu được cho trước. Trong tập dữ liệu này, có những bộ dữ liệu gây khó khăn hoặc không thể giải ra trong quá trình rút gọn chuỗi. Bên cạnh đó dữ liệu được thu nhập một cách ngẫu nhiên. Vì vậy, tập dữ liệu này không chỉ chứa số lượng lớn bộ dữ liệu kiểm tra mà còn đa dạng, phức tạp và gây khó khăn cho phương pháp xử lý chuỗi dữ liệu.

**2.2 Cấu trúc của kiểu dữ liệu đầu vào**

**Hình 1.** Cấu trúc cảu kiểu dữ liệu node

typedef struct Node\_array

{

char state[SIDE];

struct Node\_array\* parent;

int depth;

TH buoc;

struct Node\_array\* nextNode;

int value;

} Node;

Hình 1. là cấu trúc của kiểu dữ liệu node dùng để thực hiện các thao tác, hành động trên

chuỗi dữ liệu bao gồm: Một mảng chứa các kí tự tên *state,* một con trỏ chỉ tới node cha tên *parent,* số nguyên để chứa độ sâu của chuỗi dữ liệu trong thuật toán tên *depth,* một lớp để chứa bước thực hiện hành động rút gọn chuỗi dữ liệu tên *buoc,* một con trỏ để chỉ tới node tiếp theo tên *nextNode,* Số nguyên để chứa giá trị của ưu tiên hành động của chuỗi dữ liệu tên *value.*

**2.3 Depth First Search (DFS)**

**Hình 2.** Giải thuật Depth-First-Search

**DEPTH-FIRST SEARCH**

**search** (initial, goal)

**if** (initial = goal) then **return** "**Solution**"

initial.depth = 0

open = new Stack

closed = new Set

insert (open, copy(initial))

**while** (open is not empty) do

n = pop(open)

insert(closed, n)

**foreach** valid move m at n do

next = state **when** playing m at n

**if** (closed doesn't contain next) **then**

next.depth = n.depth + 1

**if** (next = goal) then **return** "**Solution**"

**if** (next.depth < maxDepth) **then**

insert (open, next)

**return** "No Solution"

Hình 2. là thuật toán depth first search được dùng để thực hiện việc rút gọn chuỗi dữ liệu được truyền vào.

Đầu tiên kiểm tra đầu vào của chuỗi dữ liệu nếu là kết quả cuối cùng mong muốn thì sẽ xuất ra kết quả và kết thúc. Khởi tạo độ sâu bằng *0* sau đó tạo 2 danh sách tên *open* và *closed* rỗng. Thêm chuỗi dữ liệu đầu vào vô trong danh sách *open* (để xét dần đến chuỗi dữ liệu đó). Tạo vòng lặp *while* với điều kiện là danh sách *open* chưa rỗng (tức vẫn còn chuỗi dữ liệu để xét). Trong vòng *while* lấy chuỗi dữ liệu từ trong danh sách open và gán nó vào danh sánh closed (tức là đã xét qua). Vòng lặp foreach với mỗi bước hành động biến đổi chuỗi dữ liệu thì gán cho bước tiếp theo *next.* Kiểm tra xem trong danh sách *closed (danh sách đã xét chuỗi dữ liệu)* có chứa bước tiếp theo *next* hay không. Nếu không thì tăng độ sâu lên 1 đơn vị để đi đến chuỗi dữ liệu tiếp theo. Sau đó kiểm tra xem bước tiếp theo *next* có phải là kết quả mong muốn sau khi rút gọn hay chưa. Nếu là kết quả cần tìm thì xuất ra kết quả và kết thúc. Còn không thì kiểm tra xem độ sâu của *node* có vẫn còn bé hơn độ sâu giới hạn hay không. Nếu còn thì thêm vào danh sách *open* chuỗi dữ liệu *next*. Kết thúc vòng lặp while (tức không giải ra được kết quả) thông báo không giải ra kết quả.

**2.4 A Star (A\*)**

**Hình 3.** Giải thuật A Star (A\*)

**A\_Star**(initial, goal)

**if** (initial = goal) then **return** “**Solution**”

initial.depth = 0

open = new Stack

closed = new Set

insert(open, copy(initial))

**do**

n = max\_valuePOP(open)

copy(next, n)

insert(closed, next)

**foreach** valid move m at n

child = Heu\_Child\_node(open)

**if** (closed doesn't contain child) **then**

**if** (child = goal) then **return** "**Solution**"

copy(n, child)

insert(open, n)

**while** open is not empty

Hình 3. là thuật toán *A Star* *(A\*)* được dùng để thực hiện việc rút gọn chuỗi dữ liệu được truyền vào.

Cũng giống như thuật toán *Depth First Search (DFS)* ở trên gần như đều giống thì khi vô vòng *do while* thì thay vì lấy ra node theo thứ tự thì ta lấy node có giá trị tốt nhất trong danh sách *open.* Hàm *max\_valuePOP* có chức năng dùng để lấy node có giá trị cao nhất ở trong danh sách *open.* Tiếp đến khi vào vòng lặp *for* thì sẽ gọi hàm *Heu\_Child\_node* để biến đổi chuỗi chữ sao cho rút gọn, đồng thời trả về giá trị *value* của node đó dựa vào 2 hàm kinh nghiệm *heuristic*.

**child.value** = **heuristic**(child) + **heuristic2**(child)

**Hình 5.** Trả về giá trị ưu tiên cho node

**Hình 4**. Trả về giá trị của node

**heuristic**(initial)

**return** prioty number of initial.buoc

**heuristic2**(initial, node)

**return** length.Initial - strlen(node.state)

Hình 4. là trả về tổng giá trị ưu tiên cho *node* để dựa vào đó thực hiện các hành động biến đổi chuỗi.

Hình 5. gồm các hàm trả về giá trị cho *node*. Hàm *heuristic* sẽ trả về giá trị ưu tiên cho bước thực hiện biến đổi chuỗi. Hàm *heuristic2* trả về giá trị độ dài của hiệu giá trị độ dài chuỗi dữ liệu trước đó trừ cho chuỗi hiện tại đang xét.

**3. KẾT QUẢ**

Sử dụng hàm srand(0) cùng với bộ dữ liệu đầu vào gồm 100 chuỗi ký tự với độ dài của chuỗi là: 2, 3, 5, 10 ký tự.

**Bảng 1**. Thời gian thực thi thuật toán DFS và Heuristic

*.*

Bảng 1 thể hiện kết quả so sánh thời gian thực thi của 2 thuật toán là: DFS và A Star (A\*). Ở đây thuật toán A\* có thời gian thấp hơn so với thuật toán DFS do tối ưu hơn ở việc ưu tiên chọn bước biến đổi chuỗi dữ liệu. Từ đó việc thực hiện biến đổi chuỗi sẽ tốn ít thời gian hơn so với thuật toán DFS.

**4. KẾT LUẬN**

Trong đồ án này chúng em đã dùng 2 giải thuật DFS và A\* để biến đổi chuỗi dữ liệu đầu vào. Giải thuật đạt được độ chính xác tương đối trên tập dữ liệu lớn về chuỗi dữ liệu. Kết quả đạt được cho thấy biến đổi chuỗi trong đồ án có thể ứng dụng để hỗ trợ rút gọn các loại dữ liệu từ word, excel và đặc biệt là rút gọn đường dẫn (link). Tuy vậy, đồ án này vẫn còn một số hạn chế cần khắc phục ở khâu giải thuật trong các đồ án tiếp theo. Để chương trình có thể rút gọn tự động thì mô hình cần phải đạt được độ chính xác cao hơn, các ràng buộc biến đổi được tốt hơn. Dùng các giải thuật, các hàm kinh nghiệm khác tối ưu hơn là hướng nghiên cứu tiếp theo của chúng em về đồ án biến đổi chuỗi dữ liệu đầu vào.

**TÀI LIỆU THAM KHẢO**

[1] <https://vi.wikipedia.org/wiki/Lỗi_tràn_bộ_nhớ_đệm>

[2]<https://www.oreilly.com/library/view/algorithms-in-a/9780596516246/ch07s02.html?fbclid=IwAR3_yiu7cLJCuwXkP_4wsCzSqfs3ZU2luLCxSSACti3Qde3yzxalEvU6rPk>