**BÀI TOÁN RÚT GỌN CHUỖI KÍ TỰ**

**Trần Hoàng Long,Nguyễn Đức Hảo**

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

**TÓM TẮT**

Các vấn đề liên quan đến mã hóa, bảo mật thông tin tạo được nhiều sức hút và mang tầm quan trọng đối với nhiều vấn đề trong thế giới hiện nay. Và một việc không hề nhỏ trong vấn đề được đưa ra là rút gọn chuỗi đưa vào theo nhu cầu của người dùng bằng các ràng buộc. Bài toán được xây dựng dựa theo cấu trúc bài toán 8-Puzzle đã được học và sử dụng 2 thuật toán là: BFS và A\*(sử dụng hàm heuristic). Bằng việc sử dụng 2 thuật toán BFS và A\* thì đã giải quyết được hầu hết các trường hợp được đưa ra và đáp ứng được nhu cầu mà người dùng đưa ra một cách tối ưu nhất.Đối với thuật toán BFS thì chỉ giải quyết được khoảng 60%-70% các trường hợp được đưa ra mặt khác thuật toán A\* thì đã khắc phục được nhưng khuyết điểm của thuật toán BFS và giải quyết một cách tối ưu nhất. Từ bài toán này chúng ta có thể phát triển nên một số phần mềm hữu ích cho nhiều lĩnh vực yêu cầu sự riêng tư về thông tin cao giúp cho sự trao đổi qua lại giữa các người dùng không bị lộ cho bên thứ 3.

**Từ khóa:** Rút gọn; mã hóa; bảo mật; ràng buộc.

1. **ĐẶT VẤN ĐỀ**

Bài toán giả quyết mỗi chuỗi kí tự ngầu nhiên gồm các kí tự {A, B, C, E} và rút gọn sao cho chỉ còn lại kí tự E (nếu giải được) hoặc in ra “No solution” nếu bài toán không thể rút gọn về kí tự E được.

1. **PHƯƠNG PHÁP:**

Đối tượng được thiết kế như là một cây tìm kiếm mà mỗi node của cây là một trạng thái được tạo ra từ trạng thái ban đầu dựa theo các ràng buộc mà đề bài đưa ra bằng 2 thuật toán đó là BFS và A\* sử dụng Heuristic. Mỗi node sẽ chứa các thông tin như: Trạng thái, độ sâu, giá trị, node tiếp theo và node cha.

typedef struct Node\_array

{

char state[SIDE];

struct Node\_array\* parent; // tree search

int depth;

Actions actions;

struct Node\_array\* nextNode; // list

} Node;

**Hình 1:** Mô hình một node.

Thuật toán BFS được thiết kế dựa theo cơ chế FIFO (first in first out) trên 2 danh sách đóng và mở. Đầu tiên các trạng thái con sẽ được thêm vào danh sách mở (danh sách chứa trạng thái sắp được duyệt) sau đó các trạng thái con sẽ được lấy ra bằng hàm FIFO\_POP để duyệt tìm các trạng thái tiếp theo rồi lại thêm vào danh sách mở nếu các trạng thái đó không nằm trong danh sách đóng.

**function** BREADTH-FIRST-SEARCH( problem,) **returns** a solution, or failure

node <— a node with STATE = problem.INITIAL-STATE, PATH-COST =0

**lf** problem.GOAL-TEST(node.STATE) **then return** SOLUTION(node)

frontier <— a FIFO queue with node as the only element

explored <— an empty set

**loop do**

**If** EMPTY?( frontier) **then** **return** failure

node <— POP( frontier) /\* chooses the shallowest node in frontier \*/

add node.STATE to ezplored

**for** **each** Actions **in** problem.ACTIONS(node.STATE) **do**

child <— CHILD-NODE( problem. node.Actions)

**lf** child.STATE Is not in explored or frontier **then**

**lf** problem.GOAL- TEST(child.STATE) **then** **return** SOLUTION(child)

frontier <— INSERT(child. frontier)

**Hình 2:** Thuật toán BFS.[1]

Hàm chuyển trạng thái sẽ xét từng kí tự của đề bài sau đó biến đổi thành chuỗi con theo các ràng buộc mà bài toán đưa ra. Ứng với từng kí tự được xét sẽ có trường hợp tương ứng (node con).

VD: Ký tự đang xét là ký tự A thì sẽ có 2 trường hợp có thể xảy ra như AB=>BC hoặc AC=>E

**Node**\* childNode(Current node,int x( Vị trí ký tự đang xét))

MakeNode child

**if** A[x]='A'

AC=>E

AB=›BC

**if** A[x]='B'

BB=>E

**if** A[x]='E'

EA=>A

EB=>B

EC=>C

EE=>E

**return** child

**Hình 3:** Hàm tìm trang thái tiếp theo.

Hàm heuristic sẽ tính giá trị ưu tiên của từng trạng thái theo độ dài và các trương hợp biến đổi có lợi giúp giải bài toán nhanh hơn.

**Int** Heuristic(Start, Current node)

Tổng =0

**For** **each** từng kí tự trong chuỗi

**IF** kí tự E hoặc AC hoặc BB

Tổng tăng 1 đơn vị

Tổng = Tổng + ( Độ dài Start – Độ dài Current node)

**Return** Tổng

**Hình 4:** Hàm Heuristic tính giá trị node.

Hàm Heuristic\_Add dùng để them một trạng thái mới vào danh sách đợi mà ở đó các trạng thái có giá trị Heuristic càng cao thì sẽ được ưu tiên lên hang đầu để đảm bảo thuật toán hoàn thành nhanh nhất có thể.

**funciton** Heuristic\_ Add(List,Start,Cunrent)

\_node< -List<-Head

**if**(List.Count<=0)

Add(Current)

**else**

**if**(Heuristic(Start,\_node)<Heuristic(Start, Cunrent))

LIFO\_ADD(Current)

**else**

FIF0\_ADD(Current)

**Hình 5:** Hàm Heuristic\_Add

Thuật toán A\* được thực hiện trên 2 danh sách đóng và mở. Danh sách mở sẽ chứa các trạng thái chờ để duyệt trong lần duyệt kế tiếp , danh sách đóng thì sẽ lưu trữ những trạng thái đã duyệt qua rồi. Từ đó ta có thể duyệt mà không sợ bị trùng lập vì mỗi lần them một trạng thái mới thì hàm A\* sẽ dò trên 2 danh sách để xem trạng thái đã xuất hiện trước đó chứ nếu chưa thì mới thêm vào danh sách mở theo hàm Heuristic\_Add.

**Function** AStart (problem)

Node <- Start.State

**If** problem.Goal\_test(node.State) **return** Solution(node);

Frontier <- Tạo danh sách mở

Explored <- Tạo danh sách đóng

**Loop** **do**

**If** Empty? (frontier) **return** false

Node <- POP(frontier)

Thêm node.state vào danh sách đóng

**Foreach** kí tự được xét trong problem.Actions(node.State) **do**

Child<- CHILD\_NODE(node, vị trí kí tự đang xét)

**If** child.State không thuộc danh sách đóng hoặc mở thì

**If** problem.Goal\_test(child.State) **return** Solution(child)

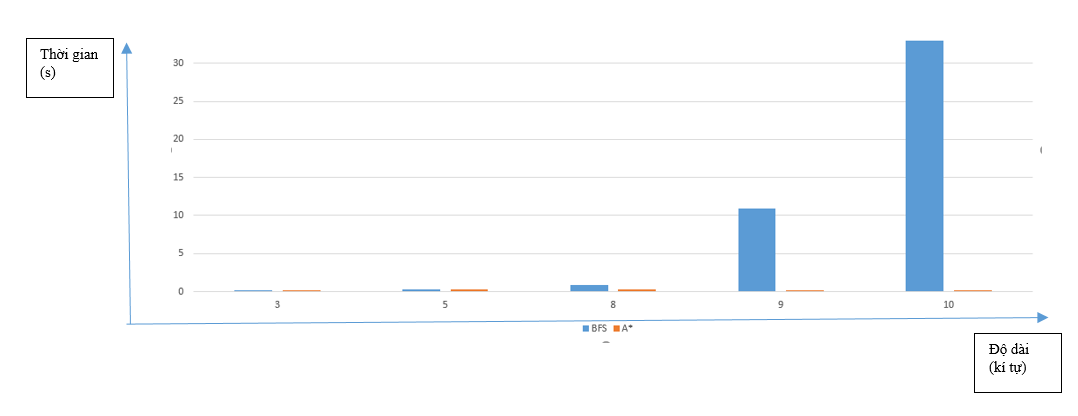
Frontier<- Heuristic\_Add(frontier, child)

**Hình 6:** Thuật toán A\*

1. **KẾT QUẢ:**

Đối với thuật toán BFS: đã giải quyết được một số đề bài nhưng vẫn còn một số hạn chế như: Thời gian chờ lâu, một số đề vẫn không tìm ra đáp án. Đối với thuật toán sử dụng hàm Heuristic: đã khắc phục được nhưng hạn chế của thuật toán DFS như: Tăng tốc độc giải quyết bài toán, tìm ra đáp án cho tất cả các đề bài được đưa ra. Dưới đây là bảng so sáng thời gian thực thi giữa 2 thuật toán với 100 bộ thử theo từng độ dài nhất định:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Độ dài  Thuật toán | 3 | 5 | 8 | 9 | 10 |
| BFS | 0.172 | 0.33 | 0.825 | 10.96 | 33+ |
| A\* | 0.163 | 0.25 | 0.321 | 0.19 | 0.233 |



**Hình 5:** Biểu đồ so sánh thuật toán BFS và thuật toán A\*

Vì thuật toán BFS vẫn còn một số hạn chế nên một số bộ thử với độ dài là 10 sẽ không tìm ra lời giải vì vậy nên nhóm chúng em xem như thời gian thực thi là rất lớn so với thuật toán A\*.

1. **KẾT LUẬN:**

**Hạn chế:** các kí tự đầu vào còn hạn chế, có ít ràng buộc, một số trường hợp vẫn có thời gian thực hiện lâu.

**Hướng phát triển tiếp tục trong tương lai:** tăng số lượng kí tự đầu vào, thêm các ràng buộc tối ưu nhất để nâng cao mức độ hiệu quả của bài toán, áp dụng các thuật toán tối ưu hơn nâng cao hiệu suất của chương trình.

**TÀI LIỆU THAM KHẢO**

[1] Slide Week\_Search.pdf page 31……………, *Huỳnh Xuân Phụng, 2020*

Baithamluan\_bentre\_Phung.pdf………………., *Huỳnh Xuân Phụng, 2019*

[AI Class] Search and Agent. <https://youtu.be/bMWfxHmIsJk>. *Huỳnh Xuân Phụng, 2020*