**VIẾT CHƯƠNG TRÌNH TỪ CHUỖI KÝ TỰ BẤT KỲ (A, B, C VÀ E), XUẤT RA MỘT DÃY CÁC BIẾN ĐỔI ĐỂ CHỈ CÒN KÝ TỰ E**

**Nhóm 2:**

*Phạm Quốc Hưng 18110128*

*Lê Phước Hưng 18110297*

**Tóm tắt**

Câu hỏi và mục đích đề tài:bắt đầu bằng một chuỗi các ký tự A, B, C và E. Viết chương trình từ chuỗi ký tự bất kỳ, xuất ra một dãy các biến đổi để chỉ còn ký tự E. Phương pháp thực hiện: viết bằng ngôn ngữ C/C++ trên Visual Studio. Kết quả: từ một chuỗi đã nhập (A, B, C và E) được biến đổi thành một ký tự E. Kết luận: vận dụng các kiến thức đã học ở môn Trí tuệ nhân tạo. Từ khoá: Breadth First Search, Best First Search, FIFO. Mô tả: Bắt đầu bằng một chuỗi các ký tự A, B, C và E (ví dụ ABABAECCEC). Chuỗi này có thể được biến đổi bằng các luật AC=E, AB=BC, BB=E và Ex=x với mọi ký tự x. Viết chương trình từ ký tự bất kỳ, xuất ra một dãy các biến đổi để chỉ còn ký tự E. Ví dụ ABBC có thể biến đổi thành AEC AC E.

**Đặt vấn đề**

Chương trình biến đổi một chuỗi ký tự sẽ là chương trình nhằm giải quyết các vấn đề về chuỗi. 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 và 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 việc có kiến thức sâu hơn về các thuật toán lập trình nên chương trình sẽ sử dụng ngôn ngữ C/C++, viết trên giao diện console.

**Phương pháp**

1. **Input**

Chương trình sẽ có đầu vào là một chuỗi ký tự, 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 thuộc khoảng input). 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. **Bảng mục báo cáo các hàm:**

|  |  |  |
| --- | --- | --- |
| 1 | list<Node\*> conAB (Node\* cha) {} | Tìm kiếm AB trong chuỗi thay bằng BC |
| 2 | list<Node\*> conAC (Node\* cha) {} | Tìm kiếm AB trong chuỗi thay bằng BC |
| 3 | list<Node\*> conBB (Node\* cha) {} | Tìm kiếm AB trong chuỗi thay bằng BC |
| 4 | list<Node\*> conEx (Node\* cha) {} | Tìm kiếm AB trong chuỗi thay bằng BC |
| 5 | void Solution (Node\* node) {} | In kết quả |
| 6 | bool inList(list<Node\*> close, Node\* n) | Kiểm tra tồn tại trong list |
| 7 | bool inQueue (priority\_queue<Node\*, vector<Node\*>, NodeCmp> pq, Node\* n) | Kiểm tra tồn tại trong priority\_queue |
| 8 | struct NodeCmp {  bool operator () (const Node\* ln, const Node\* rn) const {} } | So sánh để thêm node vào priority\_queue |
| 9 | void bestFirstSearch (Node\* root, Node\* goal) {} | Hàm xử lý theo bestfirstsearch |
| 10 | void breadthFirstSearch (Node\* root, Node\* goal) {} | Hàm xử lý theo breadFirstSearch |

1. **Các thuật toán đã sử dụng:**

***Breadth First Search:*** thuật toán này duyệt chuỗi khởi đầu theo

BFS {

Close list = []

Open list = [root]

While (open khác rỗng)

{

Node current = node đầu open

Nếu current là đích

In kết quả

Forech n in node AC (current)  
 Nếu node ko trong open và close

Thêm n vào cuối open

Forech n in node AB (current)  
 Nếu node ko trong open và close

Thêm n vào cuối open

Forech n in node BB (current)  
 Nếu node ko trong open và close

Thêm n vào cuối open

Forech n in node Ex (current)  
 Nếu node ko trong open và close

Thêm n vào cuối open

Thêm current vào close

***Best First Search:*** thuật toán Best First Search duyệt giống với thuật toán BFS dùng hàng đợi ưu tiên (so sánh độ dài chuỗi).

bestFirstSearch {

Close priority\_queue = []

Open priority\_queue = [root]

While (open khác rỗng)

{

Node current = node đầu open

Nếu current là đích

In kết quả

Forech n in node AC (current)  
 Nếu node ko trong open và close

Thêm n có độ dài ngắn nhất vào open

Forech n in node AB (current)  
 Nếu node ko trong open và close

Thêm n có độ dài ngắn nhất vào open

Forech n in node BB (current)  
 Nếu node ko trong open và close

Thêm n có độ dài ngắn nhất vào open

Forech n in node Ex (current)  
 Nếu node ko trong open và close

Thêm n có độ dài ngắn nhất vào open

Thêm current vào close

Độ phức tạp O(b^m)

**Bảng biểu:**

|  |  |  |
| --- | --- | --- |
| Mẫu thử | Breath-first Search | Best-first Search |
| ABABAEECBBACAC | 1.718 | 0.011 |
| ABAB | 0.002 | 0.001 |
| ABAB (5) | >10s | 0.786 |

**Kết luận**

Về mặt ưu điểm, 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.