TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI

KHOA CÔNG NGHỆ THÔNG TIN

BỘ MÔN MẠNG VÀ CÁC HỆ THỐNG THÔNG TIN

BÁO CÁO XÂY DỰNG CHƯƠNG TRÌNH

HỌC PHẦN TRÍ TUỆ NHÂN TẠO

1. Họ và tên: Nguyễn Ngọc Hảo

2. Mã sinh viên : 191203116

3. Lớp : CNTT5-K60

4. Nội dung báo cáo

# 4.1. Thuật toán 1

4.1.1 Giới thiệu bài toán :

Bài toán 8-puzzle (hay còn gọi là 8 số) là một bài toán quen thuộc với những người bắt đầu tiếp cận với môn Trí tuệ nhân tạo. Bài toán có nhiều phiên bản khác nhau dựa theo số ô, như 8-puzzle, 15-puzzle,… ở mức độ đơn giản nhất, em xét dạng bài toán 8-puzzle.

Bài toán gồm một bảng ô vuông kích thước 3x3, có tám ô được đánh số từ 1 tới 8 và một ô trống. Trạng thái ban đầu, các ô được sắp xếp một cách ngẫu nhiên, nhiệm vụ của người chơi là tìm cách đưa chúng về đúng thứ tự như hình dưới:

| 1 | 2 | 3 |
| --- | --- | --- |
| 8 | 0 | 4 |
| 7 | 6 | 5 |

Trong quá trình giải bài toàn, tại mỗi bước, ta giả định chỉ có ô trống là di chuyển, như vậy, tối đa ô trống có thể có 4 khả năng di chuyển (lên trên, xuống dưới, sang trái, sang phải).

4.1.2 Mô tả thuật toán :

Ở đây em dùng 2 thuật toán là BFS(Tìm kiếm theo chiều rộng) hoặc là DFS(Tìm kiếm theo chiều sâu)

Tìm kiếm theo chiều rộng(BFS) tương tự vs Tìm kiếm theo chiều sâu (DFS) chỉ khác nhau về cách duyệt nên ở đây em sẽ trình bày về DFS.

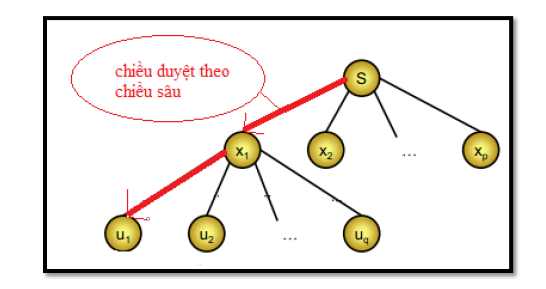
Với tư tưởng đi sâu vào từng nhánh, ta giả sử đầu vào của thuật toán là một đồ thị G = (V,E). Coi s là đỉnh gốc của V, ta sẽ bắt đầu quá trình duyệt với s.Từ s ta sẽ đi 6 thăm tới đỉnh kề với s (giả sử ở đây là u0), từ u0 ta tiếp tục quá trình duyệt với các đỉnh kề u0 (trừ các đỉnh đã thăm). Quá trình sẽ tiếp tục tới khi gặp đỉnh cần tìm hoặc đi hết nhánh thì thực hiện lùi lại đỉnh trước đó.

Xét một cách tổng quát thì khi xét một đỉnh u0 ta sẽ có hai khả năng xảy ra:

- Nếu như tồn tại đỉnh v0 kề với u0 mà chưa được thăm thì đỉnh v0 đó sẽ được đánh dấu để trở thành đỉnh đã thăm và quá trình tìm kiếm sẽ bắt đầu từ đỉnh v0 đó.

- Ngược lại, nếu mọi đỉnh kề với u0 đều đã thăm thì ta sẽ quay lại đỉnh mà trước đó ta đến đỉnh u0 để tiếp tục quá trình tìm kiếm.

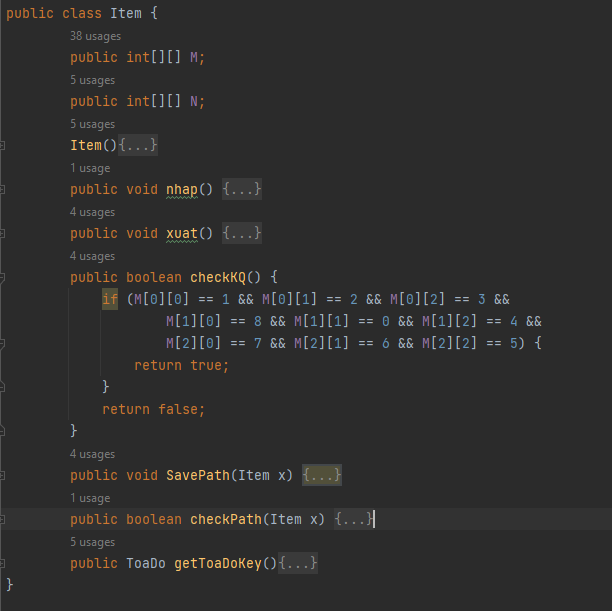
Như vậy trong quá trình thăm đỉnh bằng thuật toán tìm kiếm theo chiều sâu, ta nhận thấy ngay đỉnh càng thăm muộn thì sớm được duyệt trước (đây là cơ chế Last In First Out) . Do đó ta có thể sử dụng thủ tục đệ quy hoặc sử dụng một danh sách kiểu ngăn xếp để tổ chức cho quá trình duyệt của thuật toán. Dưới dây là minh họa cho quá trình duyệt với một cây:



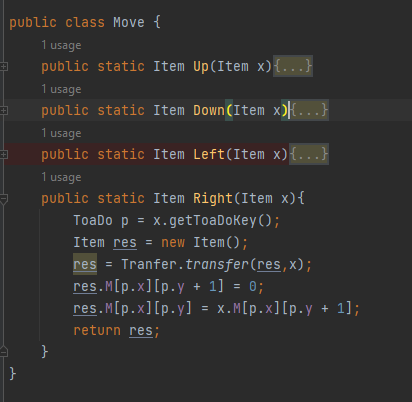
Hình trên minh họa thứ tự duyệt của thuật toán tìm kiếm theo chiều sâu: ta nhận thấy từ S sẽ thăm X1, tiếp tục tới u1 là kề với X1. Nếu u1 không phải đỉnh tìm và 7 hết nhánh thì sẽ lùi về X1 để thăm u2. Do đó quá trình duyệt sẽ là: Sx1u1u2 ... uq x2 ...

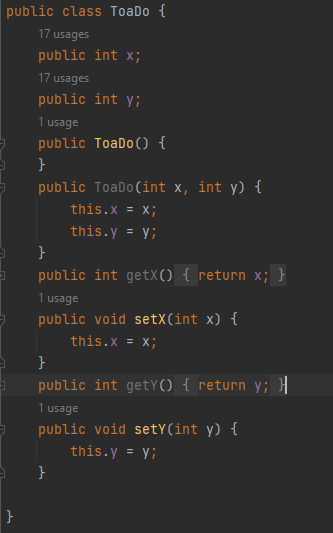
4.1.3 Kết quả cài đặt (mô tả chương trình hoặc mã chương trình)

-Xây dựng lớp Item để lưu các trạng thái và kiểm tra trạng thái kết thúc

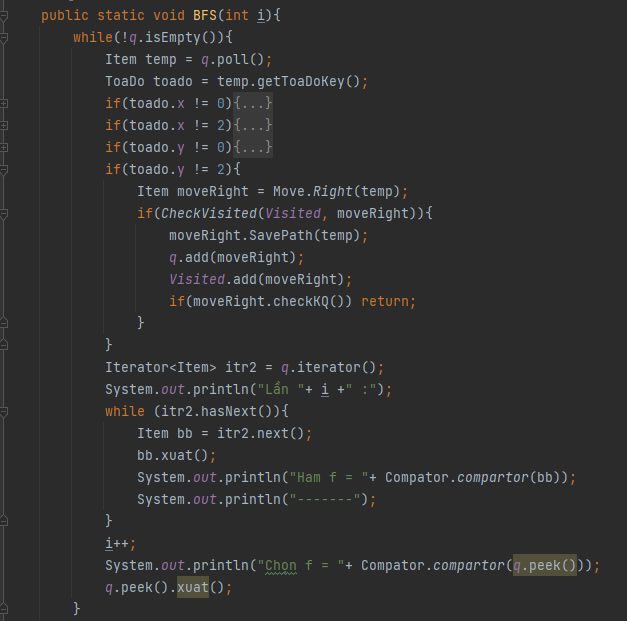


-Xây dựng lớp Move, Toado để sinh ra các trạng thái sau các lần di chuyển

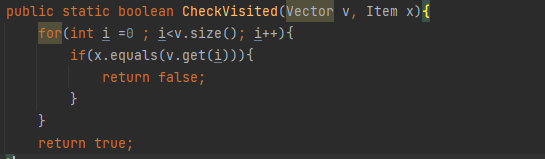




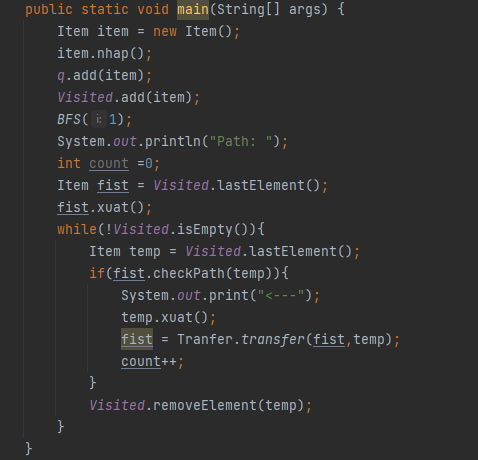
-Xây dựng phương thức BFS để giải quyết bài toán



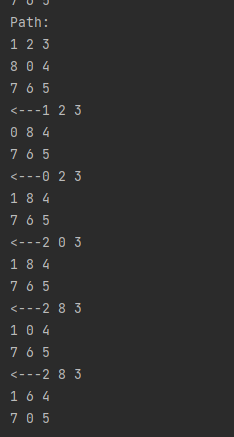
-Xây dựng phương thức checkVisied để loại bỏ các trạng thái lặp



-Xây dựng hàm Main để test và kiểm tra bài toán :



4.1.4 Kết quả thử nghiệm



# 4.2. Thuật toán 2

4.2.1 Giới thiệu bài toán

Cho *G* là tập hợp các câu tuyển:

⎤*A* ∨ ⎤*B* ∨ *P* (l)

⎤*C* ∨ ⎤*D* ∨ *P* (2)

⎤E ∨ *C* (3)

*A* (4)

*E* (5)

*D* (6)

Giả sử ta cần chứng minh *P.*

4.2.2 Mô tả thuật toán :

- Với bài toán này em sử dụng phương pháp chứng minh bác bỏ

- Thêm vào tập G câu sau : ⎤*P* (7)

| **u** | **V** | **res(u,v)** | **THOA** |
| --- | --- | --- | --- |
|  |  |  | ⎤P(7) |
| **⎤P(7)** | ⎤A∨⎤B∨P(1) | ⎤A∨⎤B(8) | ⎤A∨⎤B(8), ⎤C∨⎤D(9) |
| ⎤C∨⎤D∨P(2) | ⎤C∨⎤D(9) |
| **⎤A∨⎤B(8)** | A(4) | ⎤B(10) | C∨⎤D(9), ⎤B(10) |
| **⎤C∨⎤D(9)** | ⎤E ∨ C(3) | ⎤D∨⎤E(11) | ⎤B(10), ⎤D∨⎤E(11), ⎤C(12) |
| D(6) | ⎤C(12) |
| **⎤B(10)** | ∅ |  | ⎤D∨⎤E(11), ⎤C(12) |
| **⎤D∨⎤E(11)** | E(5) | ⎤D(13) | ⎤C(12), ⎤D(13),⎤E(14) |
| D(6) | ⎤E(14) |
| **⎤C(12)** | ⎤E ∨ C(3) | ⎤E(14) | ⎤D(13), ⎤E(14) |
| **⎤D(13)** | D(6) | • | ⎤E (14) |

Có thể hình dung vết chứng minh của bài toán trên được xác định qua thuật toán trên như sau:

• ←(13,6) ←(11,5) ← (9,3) ← (7,2)

Ta sử sử P sai, tức là ⎤*P* (7)

Áp dụng luật phân giải cho câu (7) và (2) tự được câu: ⎤*C* ∨ ⎤D (9)

Từ câu (9) và (3) ta nhận được câu: ⎤*D*∨⎤*E* (11)

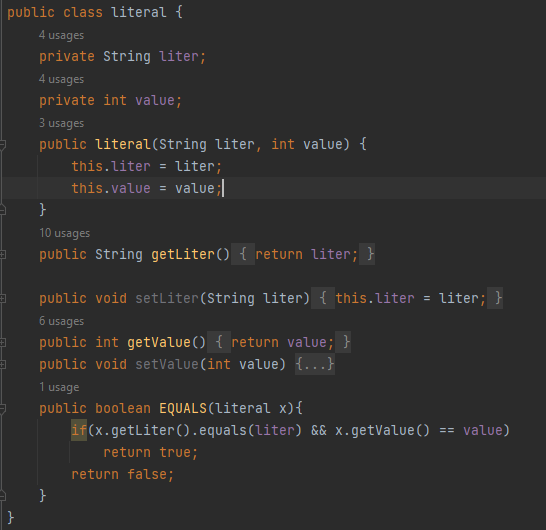
Từ câu (11) và (5) ta nhận được câu: ⎤D (13)

Từ câu (13)và (6) ta nhận được câu rỗng • hay tới đây đã xuất hiện mâu thuẫn.

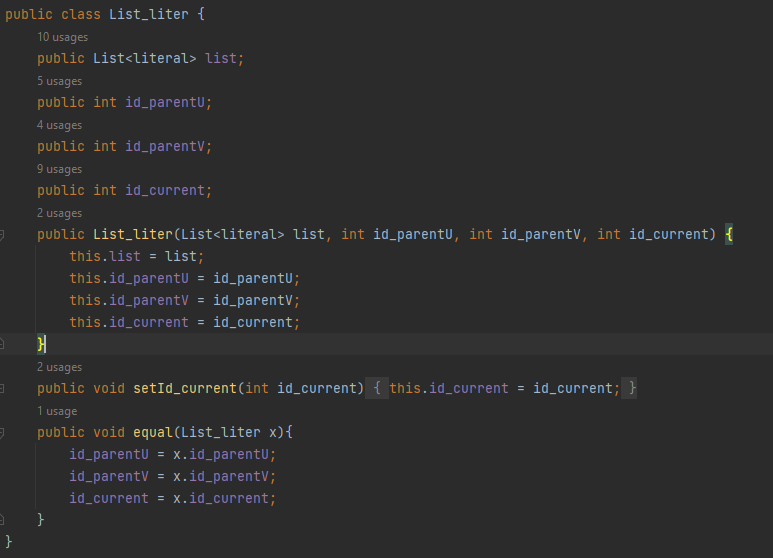
Vậy *P* là hệ quả logic của các câu (l) *--* (6). Hay P được chứng minh.

4.2.3 Kết quả cài đặt (mô tả chương trình hoặc mã chương trình)

-Xây dựng lớp literal để lưu từng literal và trạng thái của nó.Ngoài ra còn có thêm phương thức Equal để so sánh 2 literal với nhau.

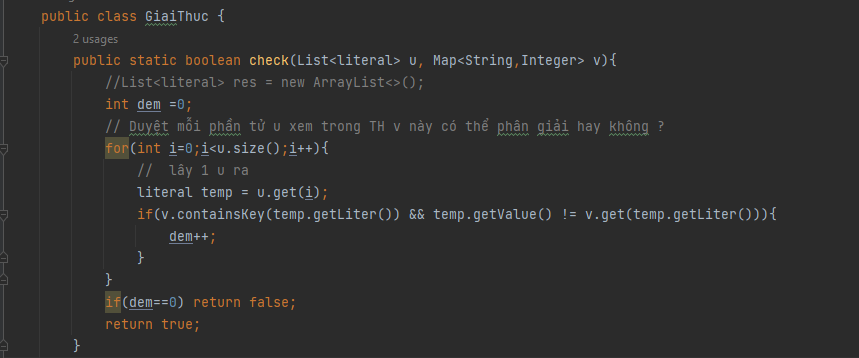


-Xây dựng lớp List-liter để lưu các câu tuyển

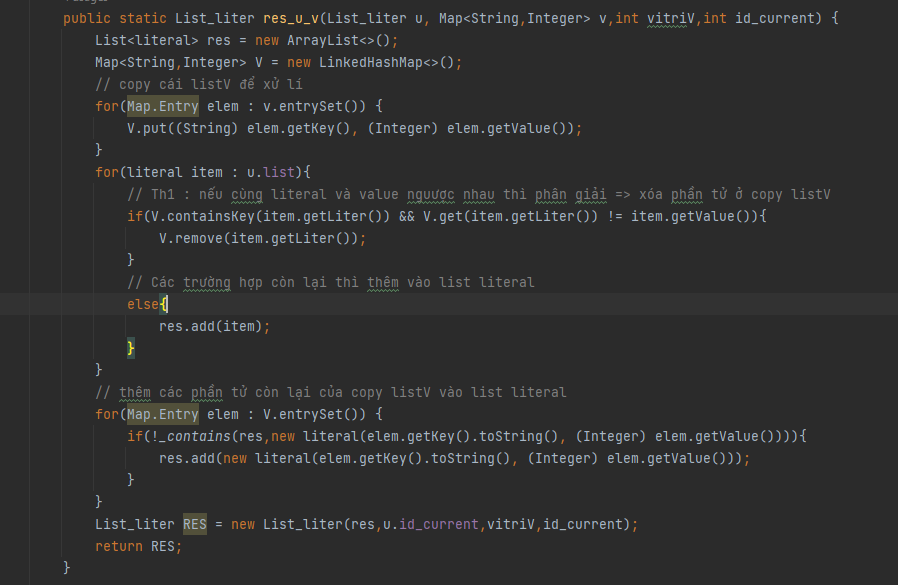


-Xây dựng lớp Giaithuc gồm có các phương thức sau :

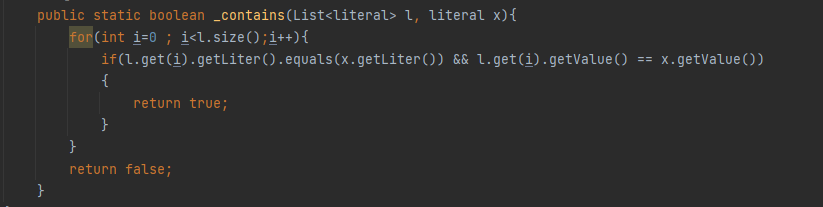
* Phương thức check để kiểm tra mỗi phần tử u xem trong TH v này có thể phân giải hay không ?



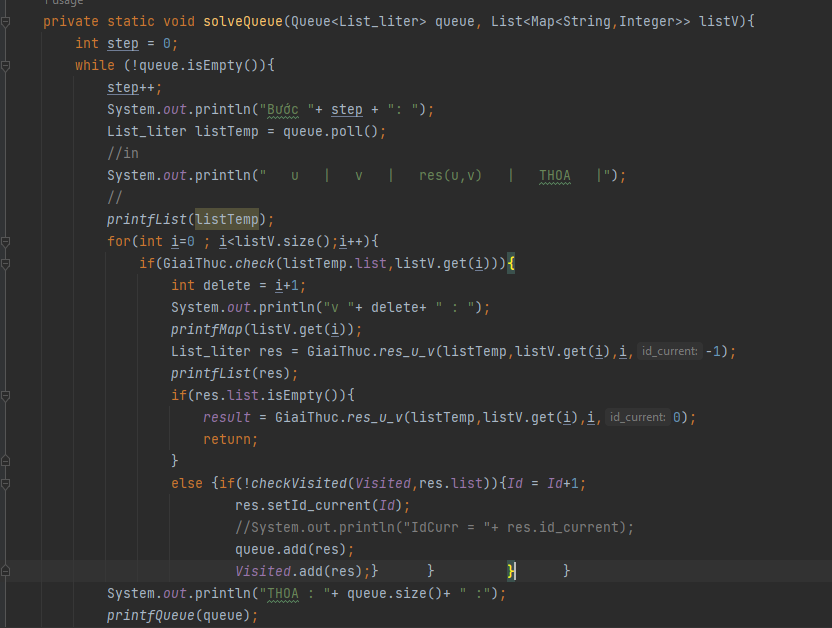
* Phương thức res\_u\_v để sinh ra kết quả sau khi phân giải u và v



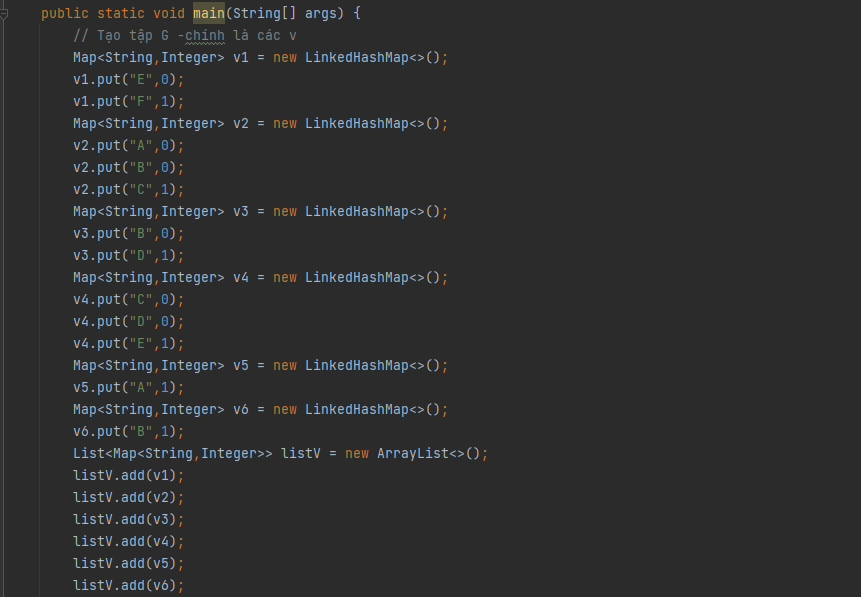
* Phương thức \_contains để kiểm tra literal X có ở trong câu tuyển không ?



-Xây dựng phương thức solveQueue để giải quyết bài toán bằng tìm kiếm theo chiều rộng và biểu diễn kết quả



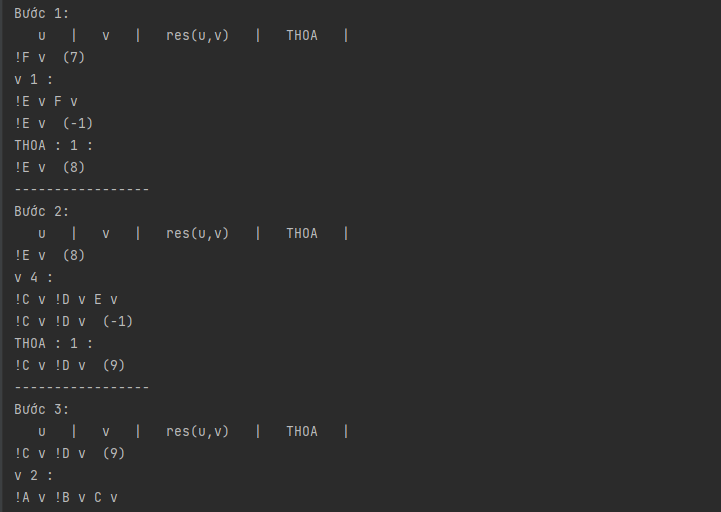
-Xây dựng đầu vào của bài toán

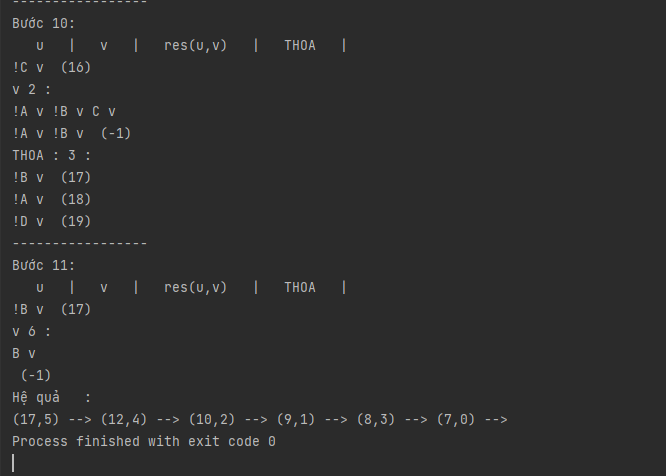


-Giải quyết bài toán bằng tìm kiếm theo chiều rộng sử dụng *hàng đợi (Queue)*



4.2.4 Kết quả thử nghiệm





# 4.3 Thuật toán 3

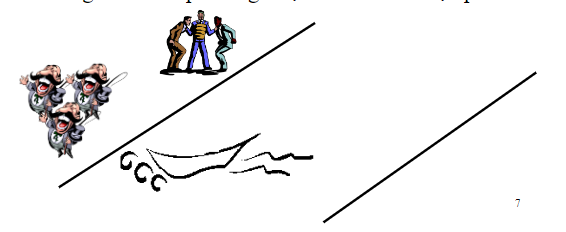
4.3.1 Giới thiệu bài toán

Xét bài toán kẻ cướp và triệu phú :

+ Có 3 triệu phú và 3 tên cướp và một chiếc thuyền ở bên bờ tả của một con sông

+ Thuyền chỉ chở được một hoặc hai người.

+ Hãy tìm cách đưa 3 triệu phú sang sông sao cho ở mỗi bờ sông số kẻ cướp không được lớn hơn số triệu phú



4.3.2 Mô tả thuật toán

Dùng một bộ 3 số (a,b,k) để biểu diễn các trạng thái

a - số triệu phú, b - số kẻ cướp ở bờ tả

k=1 thuyền đang ở bờ tả, ngược lại k=0

Không gian trạng thái được xác định như sau :

Trạng thái đầu (3,3,1) và trạng thái kết thúc (0,0,0)

Tập các toán tử bao gồm các toán tử:

+Thuyền chở 1 kẻ cướp

+Thuyền chở 1 triệu phú

+Thuyền chở 1 triệu phú, 1 kẻ cướp

+Thuyền chở 2 kẻ cướp

+Thuyền chở 2 triệu phú

Với bài toán này em sử dụng thuật toán tìm kiếm mù theo chiều rộng và có loại bỏ các trạng thái lặp

Mô tả quá trình tìm kiếm lời giải cho bài toán :

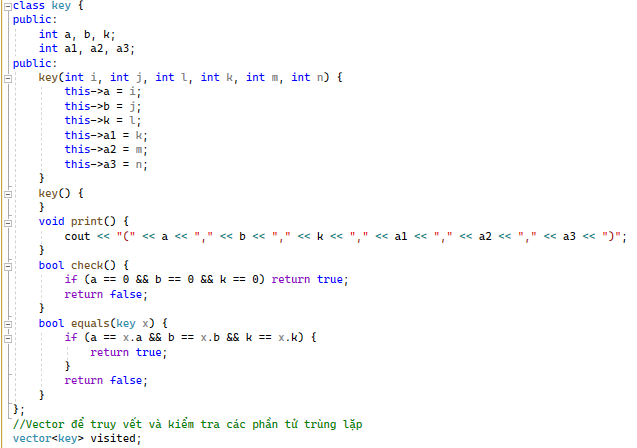
| **u** | **Trạng thái kề v** | **Danh sách L (Hàng đợi)** |
| --- | --- | --- |
|  |  | (3,3,1) |
| (3,3,1) | (3,2,0),(2,3,0),(2,2,0),(3,1,0),(1,3,0) | (3,2,0),(2,2,0),(3,1,0) |
| (3,2,0) | (3,3,1) | (2,2,0),(3,1,0) |
| (2,2,0) | (3,2,1),(2,3,1),(3,3,1) | (3,1,0), (3,2,1) |
| (3,1,0) | (3,2,1) | (3,2,1) |
| (3,2,1) | (3,1,0),(2,2,0),(2,1,0),(3,0,0),(1,2,0) | (2,2,0),(3,0,0) |
| (2,2,0) | (2,3,1),(3,2,1),(3,3,1) | (3,0,0) |
| (3,0,0) | (3,1,1),(3,2,1) | (3,1,1) |
| (3,1,1) | (3,0,0),(2,1,0),(2,0,0),(1,1,0) | (1,1,0) |
| (1,1,0) | (1,2,1),(2,1,1),(2,2,1),(1,3,1),(3,1,1) | (2,2,1) |
| (2,2,1) | (2,1,0),(1,2,0),(1,1,0),(2,0,0),(0,2,0) | (0,2,0) |
| (0,2,0) | (0,3,1),(1,2,1),(1,3,1),(2,2,1) | (0,3,1) |
| (0,3,1) | (0,2,0),(0,1,0) | (0,1,0) |
| (0,1,0) | (0,2,1),(1,1,1),(1,2,1),(0,3,1),(2,1,1) | (0,2,1),(1,1,1) |
| (0,2,1) | (0,1,0),(0,0,0) | (1,1,1),(0,0,0) |
| (1,1,1) | (1,0,0),(0,1,0),(0,0,0) | (0,0,0) |
| (0,0,0) | **Trạng thái kết thúc – Dừng** |  |

Thuật toán dừng lại khi gặp trạng thái kết thúc (0,0,0)

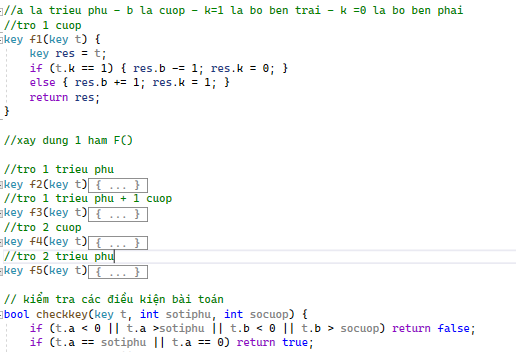
4.3.3 Kết quả cài đặt

Với bài toán này em dùng ngôn ngữ C++ để giải quyết

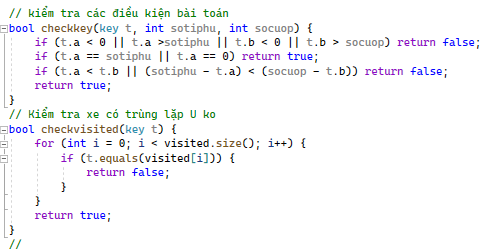
-Xây dựng lớp Key để lưu các trạng thái của bài toán .Trong lớp này có thêm các phương thức *check -* kiểm tra xem trạng thái này có phải là trạng thái kết thúc không? , phương thức *equals -* kiểm tra 2 trạng thái có giống nhau không ?



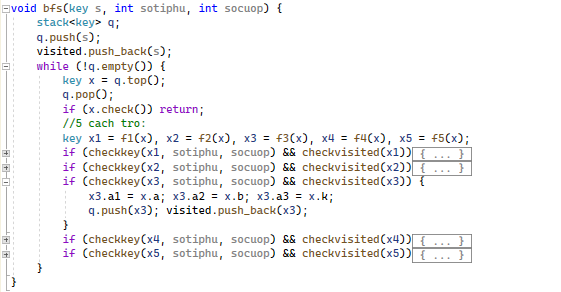
-Xây dựng các toán tử để sinh ra các trạng thái (đưa 1 cướp, 2 cướp ,... qua sông)



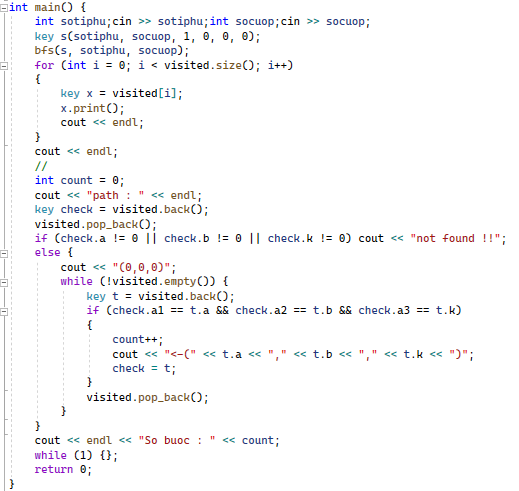
-Xây dựng các hàm *checkkey* - kiểm tra trạng thái này hợp lệ với yêu cầu bài toán không ? , hàm checkVisited để kiểm tra các trạng thái có lặp nhau không ?



-Xây dựng thuật toán tìm kiếm mù theo chiều sâu bằng cách sử dụng *Ngăn Xếp(Stack)*

**

-Xây dựng hàm main để giải quyết bài toán và biểu diễn bài toán



4.3.4 Kết quả thử nghiệm

Với đầu vào là nhập số tỷ phú và số tên cướp :

