Trường đại học Khoa học Tự nhiên Khoa Công nghệ thông tin

BÁO CÁO ĐỒ ÁN 1

TÌM KIẾM HEURISTIC VỚI A*

Môn học: *Cơ sở trí tuệ nhân tạo*

Lớp: *16CNTN*

Sinh viên thực hiện: *Hoàng Thiên Nữ - Dương Nguyễn Thái Bảo*

Tháng 10 năm 2018

I. Thông tin thành viên

Họ và tên	**MSSV**	**Email**
Hoàng Thiên Nữ	1612880	1612880@student.hcmus.edu.vn
Dương Nguyễn Thái Bảo	1612840	1612840@student.hcmus.edu.vn

II. Phân công công việc

Công việc	**Phụ trách**	**Mức độ hoàn thành**
*Cài đặt A**	Bảo	100%
*Cài đặt ARA**	Nữ	100%
Sinh test	Nữ	100%
Cài đặt GUI	Bảo	100%
Chứng minh heuristic chấp nhận	Nữ	100%
Sơ đồ hệ thống phần mềm	Bảo	100%
Tổng hợp báo cáo	Nữ	100%

III. Mức độ hoàn thành đồ án

Yêu cầu	**Mức độ hoàn thành**
Yêu cầu 1	100 %
Yêu cầu 2	100 %
Toàn bộ đồ án	100 %

IV. Yêu cầu 1

1. Cấu trúc dữ liệu cài đặt

Cấu trúc dữ liệu đầu vào gồm:

- S: tọa độ điểm bắt đầu.
- G: tọa độ điểm đến.
- map_mat: bản đồ được lưu dưới dạng một ma trận vuông N x N.
- Cấu trúc dữ liệu phần xử lý:
- h(s): giá trị heuristic của s.
- g(s): chi phí đường đi từ S đến s.
- pre: lưu vết đường đi.
- OPEN: có cấu trúc là PriorityQueue lưu giá trị f = g + h.

2. Thuật toán

Thuật toán sử dụng để giải quyết là A* với hàm heuristic sử dụng là khoảng cách Euclid của các điểm đến điểm Goal.

Thuật toán được mô tả dưới đoạn mã giả sau:

```
Đẩy S với f(S) = h(S) vào OPEN.
Lặp cho đến khi OPEN rỗng:
     Lấy s có f(s) nhỏ nhất ra khỏi OPEN.
     Duyệt các đỉnh s’ kề với s:
          Nếu g(s’) > g(s) + c(s, s’) thì:
               Cập nhật lại g(s’) = g(s) + c(s, s’)
               Đẩy u vào Heap với f(s’) = g(s’) +
               h(s’)
```

3. Bộ test

Bộ test 1

input	output
19 0 0 18 18 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0	30 (0,0) (1,1) (2,2) (3,3) (4,4) (5,5) (6,6) (7,7) (8,8) (9,8) (10,7) (11,7) (12,7) (13,8) (13,9) (12,10) (12,11) (11,12) (10,13) (9,14) (9,15) (10,16) (11,17) (12,18) (13,18) (14,18) (15,18) (16,18) (17,18) (18,18) S----------------- -x----oooo----oooo- --x---ooo-----oo--- ---x-oo-------o---- ----x--------o-o--- --oo-x-------ooo--- -ooo--x-----ooo---- -------x--ooooo-oo- --------xooo------- ---o----xoo---xxo-- -o-o---xoo---xoox-- -o-o---xo---x--o-x- -o-o---xo-xx---o--x

0 1 0 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 1 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0	-------oxxooo--o--x -o-o-----oooo--o--x ---------------o--x ---------------o-ox ---------------o-ox --------------oo--G

Bộ test 2:

input	output
10 0 0 9 9 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 1 0 0 1 1 0 0 1 0 1 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 0 0 1 0 0 1 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 0	-1

Bộ test 3:

input	output
10 0 0 7 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0	11 (0,0) (1,1) (2,2) (3,3) (4,4) (5,5) (6,4) (6,3) (7,2) (7,1) (7,0) S---oo---- -xoooo----

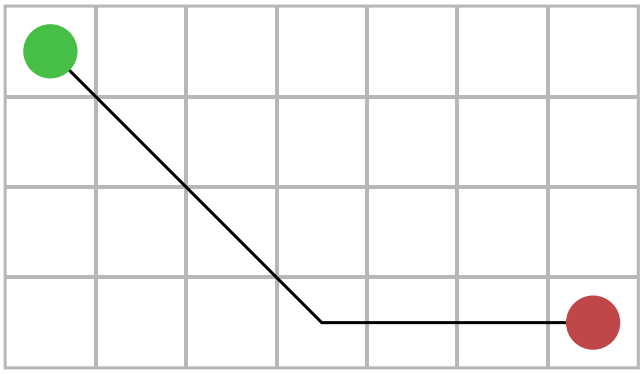
0 1 0 0 0 0 1 0 0 0 0 1 0 1 0 1 0 0 1 0 1 1 1 1 1 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 1 1 1 0 0 0 1 0 0 1 1 1 0 0	--x------- -o-x--o--- -o-oxo--o- ooooox---- --oxx-o-o- Gxxo-o---- -oo---ooo- --o--ooo--

V. Yêu cầu 2

1. Hàm heuristic đề xuất

Hàm heuristic được đề xuất ở đây là khoảng cách chéo của các điểm trên tọa độ đến Goal.

$$h(A,B) = \max(|A.x - B.x|, |A.y - B.y|)$$



Hàm heuristic được lập dựa trên ý tưởng như sau: vì ta có thể di chuyển chéo, nên ta sẽ ưu tiên việc di chuyển chéo nếu cần thiết. Bởi 1 bước chi chuyển chéo sẽ tối ưu hơn việc di chuyển 2 bước (1 bước dọc và 1 bước ngang). Sau khi di chuyển chéo, ta cần phải cộng thêm một lượng Δ nào đó để có thể đi đến G. Do đó, ở đây ta cần lấy ta cần phải lấy giá trị lớn nhất như công thức được nêu.

Đây là một hàm heuristic chấp nhận. Để chứng minh một hàm heuristic chấp nhận, ta cần chứng minh giá trị hàm heuristic của mỗi điểm không vượt quá chi phí thực. Dễ thấy, trong trường hợp thuận lợi nhất (bản đồ không có chướng ngại vật), đây là lời giải tối ưu nhất. Do vậy $h(x) \leq h^*(x)$.

2. Thuật toán ARA*

ARA* sử dụng hàm heuristic có trọng số, cụ thể là:

$$f(x) = g(x) + \varepsilon * h(x) \ (\varepsilon \geq 1)$$

Ý nghĩa của hệ số khi nhân vào là làm cho việc mở rộng các nút ít hơn và như vậy thuật toán sẽ chạy nhanh hơn. Tuy nhiên, việc nhân vào hệ số cho hàm heuristic sẽ có thể vi phạm tính chất "chấp nhận" của hàm. Do vậy, lời giải có thể sẽ không tối ưu.

Bản chất của ARA* chính là chạy A* nhiều lần trong khoảng thời gian cho phép. Đầu tiên, thuật toán chạy A* với trọng số ε lớn rồi sau đó giảm xuống cho đến khi $\varepsilon = 1$. Thuật toán ARA* sẽ mở rộng nhiều nút hơn A* để có thể tìm được lời giải tốt nhất.

Nếu ta chạy A* nhiều lần và giảm hệ số thì thuật toán sẽ không hiệu quả. Do đó, ARA* phải sử dụng lại kết quả của A* chạy lần trước đó. Chương trình có thể mô tả bởi đoạn mã giả sau:

```
Hàm ImprovePath:
Trong khi fvalue(s_goal) > min(fvalue(s), s ∈ OPEN):
    Loại s khỏi tập OPEN
    CLOSED = CLOSED ∪ {s}
    Với mỗi s' kề với s
        Nếu s' chưa được viếng năm thì g(s') = oo
        Nếu g(s') > g(s) + c(s, s')
            g(s') = g(s) + c(s, s')
            Nếu s' không thuộc CLOSED
                Thêm s' vào OPEN với fvalue(s')
            Ngược lại
                Thêm s' vào INCONS
```

Hàm ARA_STAR:

Khởi tạo $g(s_{goal}) = \infty, g(s_{start}) = 0$

OPEN = CLOSED = INCONS = 0

Thêm s_{start} vào tập OPEN với $fvalue = h(s_{start})$

Gọi hàm ImprovePath()

$$\epsilon' = \min(\varepsilon, \frac{g(s_{goal})}{\min_{s \in OPEN \cup INCONS} g(s) + h(s)})$$

Xuất lời giải với ϵ hiện tại

Trong khi $\varepsilon > 1$

    Giảm ε

    Di chuyển các đỉnh từ INCONS sang OPEN

    Cập nhật lại priority queue OPEN cho tất cả các đỉnh s thuộc OPEN

    Gán CLOSED = 0

    ImprovePath()

Hàm ImprovePath() sẽ tính toán lời giải với các ε khác nhau. Sau đó, thuật toán ARA* sẽ lần lượt gọi hàm ImprovePath và giảm ε.

Ở đây ta có một khái niệm mới "bất nhất quán" (inconsistency). Một trạng thái được gọi là bất nhất quán khi mà giá trị g của nó bị giảm và trạng thái sẽ được mở rộng ở lần tiếp theo. Khi s được mở rộng, giá trị g sẽ được tính lại bởi giá trị g của cha s. Như vậy, sự bất nhất quán sẽ được lan ra đến các con của s thông qua việc mở các nút. Cuối cùng, các trạng thái con sẽ không phụ thuộc vào s, vì giá trị g của chúng thể tối ưu được. Vì thế nên chúng cũng không được thêm vào OPEN. OPEN sẽ gồm những trạng thái bất nhất quán: nếu giá trị g của trạng thái có thể được tối ưu thì sẽ được thêm vào OPEN, và khi chúng được mở rộng thì sẽ được loại khỏi OPEN cho đến khi giá trị g của chúng được tối ưu ở lần tiếp theo. Do đó, OPEN là danh sách ta có thể xem được những trạng trạng cần được lan truyền sự bất nhất quán. Thuật toán A* sử dụng hàm heuristic nhất quán đảm bảo rằng mỗi trạng thái được mở rộng ít nhất 1 lần. Tuy nhiên, với epsilon > 1 có thể làm vi phạm tính

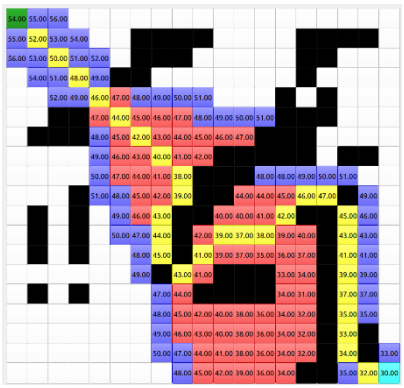
nhất quán. Do vậy, kết quả tìm kiếm A* có thể mở rộng các trạng thái nhiều lần. Vì thế ta cần phải đảm bảo mỗi trạng thái chỉ được mở rộng không quá 1 lần, mà epsilon vẫn được giữ. Ý tưởng đó được thực hiện việc ta chỉ cần thêm vào OPEN những trạng thái có giá trị g có thể được tối ưu khi trạng thái đó chưa được mở rộng. Tập các trạng thái đã được mở rộng sẽ được lưu ở CLOSED. Với cách làm đó, ta sẽ đảm bảo được mỗi trạng thái sẽ được mở rộng nhiều nhất 1 lần. Nhưng khi đó, OPEN sẽ không còn chưa những trạng thái bất nhất quán nữa. Thực tế, nó sẽ chứa những trạng thái bất nhất quán mà chưa được mở rộng. Và để sử dụng lại kết quả cho lần duyệt tiếp theo, ta giữ một biết INCONS để lưu những trạng thái bất nhất quán mà không thuộc OPEN. Vậy INCONS và OPEN giữ tất cả các trạng thái bất nhất quán. Ta có thể dùng lại chúng để lan truyền cho lần duyệt tiếp theo.

Sự khác biệt giữa ImprovePath và A* là điều kiện dừng. Bởi vì ImprovePath sử dụng lại kết quả tìm kiếm ở lần trước đó, nên s_{goal} có thể là không nhất quán và không được thêm vào OPEN. Điều kiện dừng của A* khi đó sẽ sai. Mặc khác, tìm kiếm A* có thể dừng ngay khi $f(s_{goal})$ bằng với giá trị f của các trạng thái thuộc OPEN. Điều kiện được dùng trong ImprovePath sẽ tránh được việc mở rộng s_{goal} khi các trạng thái khác có cùng giá trị f.

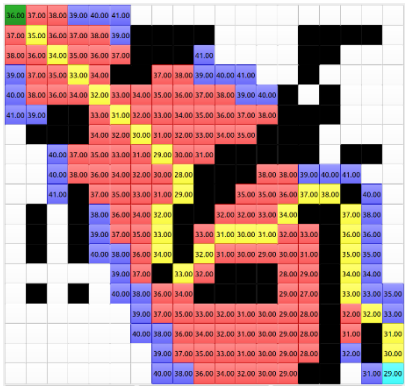
Ở phần chính của thuật toán ARA*, ta lần lượt gọi ImprovePath với giá trị ε giảm dần. Trước khi gọi hàm ImprovePath, danh sánh trong OPEN sẽ được xây dựng lại bằng cách sao chép dữ liệu các đỉnh ở trong INCONS. Vì OPEN đã được sắp xếp với giá trị f hiện tại, sau khi gọi ImprovePath ta sẽ có được lời giải tối ưu với ε tương ứng.

Ta có thể tính được giới hạn tối ưu bằng tỉ lệ giữa $g(s_{goal})$ (giới hạn trên của chi phí lời giải tối ưu) và giá trị f nhỏ nhất trong tập các trạng thái bất nhất quán (giới hạn dưới của chi phí lời giải tối ưu). Nếu giá trị $g(s_{goal})$ bằng với lời giải tối ưu thì giá trị giới hạn tối ưu sẽ nhỏ hơn 1. Có nghĩa là khi đó ta đã tìm được lời giải tối ưu. Giới hạn tối ưu thật sự được tính bằng cách lấy min của ε và tỉ lệ như trên. Ban đầu, giá trị này được cho là dùng để tính toán việc chúng ta sẽ giảm ε như thế nào. Tuy nhiên, nhiều thử nghiệm cho thấy rằng việc giảm ε qua từng bước càng nhỏ càng tốt. Lý do đơn giản là vì việc giảm một lượng nhỏ ε sẽ giúp ta tìm được lời giải tối ưu sớm hơn. Bởi vì khi giảm một lượng lớn epsilon có thể gây nên việc có quá nhiều trạng thái được mở trong lần tìm kiếm tiếp theo.

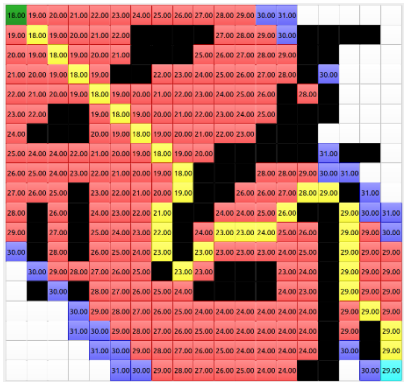
3. Kết quả với giá trị ε khác nhau



Epsilon = 3



Epsilon = 2



Epsilon = 1

Ta có thể thấy, với $\varepsilon = 3$, số trạng thái được mở là ít nhất, do vậy kết quả bài toán chưa thực sự tối ưu. Đối với $\varepsilon = 2$, số trạng thái được mở là nhiều hơn và lời giải trong trường hợp này tối ưu hơn. Số trạng thái được mở tại $\varepsilon = 1$ là nhiều nhất, tuy nhiên ở đây vì lời giải trong $\varepsilon = 2$ là tối ưu nên kết quả không có gì thay đổi.

4. Bộ test

Chạy chương trình bằng terminal với dòng lệnh sau:

$$python3\ ara\_start.py\ < input\_file > < output\_file > < time\_limit > \\ < epsilon >$$

File output với định dạng:

- Dòng đầu tiên là thời gian chạy của để tìm ra lời giải.
- Tiếp theo là giá trị epsilon cuối cùng.
- Những dòng tiếp theo là lời giải tối ưu tìm được trong khoảng thời gian cho phép được định dạng giống với yêu cầu 1.

Bộ test 1: time limit = 0.01 và epsilon = 3

input	output
15 0 0 9 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0	Time: 0.011434000000000041 Epsilon: 1.7272727272727273 20 (0,0) (1,1) (2,1) (3,2) (3,3) (3,4) (4,5) (5,6) (6,7) (7,8) (7,9) (6,10) (5,11) (5,12) (6,13) (7,13) (8,13) (9,12) (9,11) (9,10) S---------- -x-o---------- -xooooooooooo--

0 0 1 1 1 1 0 0 1 0 0 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 0 1 0 1 0 0 1 0 1 1 0 0 0 0 1 0 1 0 0 0 0 1 1 1 0 1 0 1 1 1 0 1 0 0 1 1 0 0 0 1 0 0 1 0 1 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 1 1 1 0 0 0	--xxx---------- -----x---ooooo- --o---x-o--xx-- --oooo-xo-xoox- ---ooo--xx-oox- --------ooooooxo -o-o--o-ooGxx-o -o----ooo-o-ooo -o--oo---o--o-o ---o-----oo---- --o----------- ---o-ooo-ooo---

Bộ test 2: time limit = 0.01 và epsilon = 5

input	output
8 0 0 6 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 0 1 1 0 0 0 0 1 0 1 1 0 0 0 0 1 0 1 0 0 0 0 0 1 1 1 0 0	Time: 0.0009149999999999991 -1

Bộ test 3: time limit = 0.05 và epsilon = 1

input	output
15 0 0 9 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 1 0 0 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 0 0	Time: 0.011613999999999979 Epsilon: 1.0 16 (0,0) (1,1) (2,1) (3,2) (4,2) (5,1) (6,1) (7,2) (8,3) (9,4) (10,5) (11,6) (11,7) (11,8) (10,9) (9,10) S---------- -x-o---------- -xooooooooooo-- --x---------- --x------ooooo-

0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 0 1 0 1 0 0 1 0 1 1 0 0 0 0 1 0 1 0 0 0 0 1 1 1 0 1 0 1 1 1 0 1 0 0 1 1 0 0 0 1 0 0 1 0 1 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 1 1 1 0 0 0	-xo-----o------ -xoooo--o--oo-- --xooo-----oo-- ---x----ooooo-o -o-ox-o-ooG---o -o---xoooxo-ooo -o--ooxxxo--o-o ---o-----oo---- --o---------- ---o-ooo-ooo---

VI. Sơ đồ hệ thống phần mềm

Chương trình gồm 4 lớp đối tượng được liệt kê từ mức nhỏ tới lớn với các chức năng chính tương ứng như sau:

Lớp	Chức năng chính	Thuộc tính chính	Phương thức chính
Node	Mô tả trạng thái của mỗi nút trong đồ thị, cũng như mỗi ô trên bảng.	id (tọa độ trên bảng). Giá trị g, h. Parent: Nút tiền nhiệm trên đường đi từ Start tới nút hiện tại. Btn: đối tượng đồ họa (1 nút bấm) tương ứng với vị trí của nút trên bảng.	getF(eps): tính giá trị $f = g + \varepsilon * h.$ setOpened()/setClosed(), setOnPath(): cập nhật trạng thái đóng/mở, trên đường đi của nút trong quá trình chạy thuật toán. bindButton(btn): liên kết với button tương ứng với tọa độ của nút trên bảng của giao diện đồ họa. Khi trạng thái của nút thay đổi thì màu sắc hoặc nội dung của button cũng thay đổi theo.
Graph	Mô tả đồ thị, lưu trữ các nút, đọc đồ thị từ file, ghi đồ thị vào file.	N: kích thước bảng. nodes: tập hợp các đỉnh trong đồ thị.	read(filename): đọc đồ thị từ file. save(filename): lưu đồ thị xuống file. getStart(), getGoal(): tìm đỉnh Start, Goal.

Solver	Cài đặt thuật toán ARA* và điều khiển quá trình chạy thuật toán.	graph: đồ thị dùng để chạy thuật toán. sleepTime: thời gian delay trên giao diện để người dùng có thể quan sát quá trình chạy của thuật toán. eps: giá trị epsilon bắt đầu. time: thời gian giới hạn chạy thuật toán. heuristic: hàm Heuristic sử dụng trong thuật toán, có thể là các hàm Const, L_1, L_2, L_{oo}.	const, L1, L2, Loo: các hàm Heuristic có thể lựa chọn. bindGraph(graph, app): liên kết một đồ thị với giao diện chương trình để thực hiện các cập nhật trên giao diện khi trạng thái nút thay đổi. solve(), stop(), reset(): các hàm điều khiển quá trình chạy. ara_star(): hàm cài đặt thuật toán ARA*. publishSolution(): công bố đường đi vừa tìm được.
Main	Thể hiện giao diện tương tác người dùng của chương trình.	Các đối tượng giao diện đồ họa. Phần chính trên giao diện là bảng gồm NxN nút bấm biểu diễn đồ thị dạng lưới, bên dưới là các nút điều khiển, trạng thái,... solver: đối tượng dùng để chạy thuật toán ARA*. graph: đồ thị tương ứng với bảng, dùng để chạy thuật toán ARA*.	Các hàm thiết lập giao diện. Các phương thức xử lý sự kiện như: • Mở file và đọc đồ thị khi bấm nút “Load graph”. • Lưu đồ thị xuống file khi bấm nút “Save file”. Chạy/dừng/khởi tạo lại thuật toán khi bấm nút “Solve”/”Stop”/”Reset”. Các hàm cập nhật giao diện như thay đổi màu nút trên bảng khi đỉnh tương ứng trên đồ thị thay đổi trạng thái, cập nhật thời gian chạy, …

VII. Tài liệu tham khảo

1. Maxim Likhachev, Geoff Gordon and Sebastian Thrun, "ARA*: Anytime A* with Provable Bounds on Sub-Optimality," Advances in Neural Information Processing Systems 16 (NIPS), MIT Press, Cambridge,MA, 2004