ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

Báo cáo Project 1

ARES'S ADVENTURE

MÔN CƠ SỞ TRÍ TUỆ NHÂN TẠO

Giảng viên hướng dẫn: Thầy Lê Hoài Bắc

Thầy Nguyễn Thanh Tình

Nhóm sinh viên thực hiện:

22120039	Nguyễn Tuấn Công
22120068	Nguyễn Anh Đức
22120109	Phạm Ngọc Hòa
22120460	Dương Hoài Minh

THÀNH PHỐ HỒ CHÍ MINH, THÁNG 11/2024

LỜI NÓI ĐẦU

Bài báo cáo **Project 1 - Ares's Adventure** trình bày việc áp dụng các thuật toán tìm kiếm: *Breadth-First Search*, *Depth-First Search*, *Uniform Cost Search*, và *A* Search* trong môn học Cơ sở trí tuệ nhân tạo để xây dựng chương trình giải quyết bài toán mê cung phức tạp. Đây là một thách thức thú vị, đòi hỏi sự hiểu biết sâu sắc về cách hoạt động của từng thuật toán, cũng như khả năng ứng dụng linh hoạt để đạt được hiệu quả tốt nhất cho từng tình huống trong trò chơi.

Chúng em chân thành gửi lời cảm ơn đến **Thầy Lê Hoài Bắc**, **Thầy Nguyễn Thanh Tình**, **Thầy Nguyễn Ngọc Đức** và **Thầy Nguyễn Trần Duy Minh**, các giảng viên đã dày công truyền đạt cho chúng em những kiến thức lẫn kinh nghiệm vô cùng quý giá trong bộ môn Cơ sở trí tuệ nhân tạo. Chính sự hỗ trợ tận tâm của các thầy đã giúp chúng em có được nền tảng vững chắc để thực hiện dự án này.

Chúng em kính mong nhận được những đánh giá và nhận xét từ thầy để có thể học hỏi, rút kinh nghiệm và hoàn thiện hơn trong những dự án tương lai.

MỤC LỤC

1 Phân chia công việc

Thành viên	MSSV	Mô tả công việc	Đánh giá
Nguyễn Tuấn Công	22120039	Thuật toán A* + Trình bày Video Demo	Hoàn thành 100%
Nguyễn Anh Đức	22120068	Thuật toán DFS + Trình bày báo cáo	Hoàn thành 100%
Phạm Ngọc Hòa	22120109	Thuật toán BFS + Thiết kế UI	Hoàn thành 100%
Dương Hoài Minh	22120460	Thuật toán UCS + Thiết kế UI	Hoàn thành 100%

2 Đánh giá mức độ hoàn thành yêu cầu dự án

- **Input**

  - 10 mê cung đầu vào theo đúng tiêu chuẩn với các thông số đa dạng: số lượng đá, khối lượng mỗi đá, cấu trúc mê cung,...

- **Output**

  - 10 kết quả đầu ra theo đúng tiêu chuẩn với các thông số: tên thuật toán sử dụng, Steps - số bước đi, Weight - tổng khối lượng đá đẩy, Node - tổng số node thuật toán đã duyệt, Time - thời gian sử dụng, Memory - bộ nhớ sử dụng, hướng dẫn các bước đi.

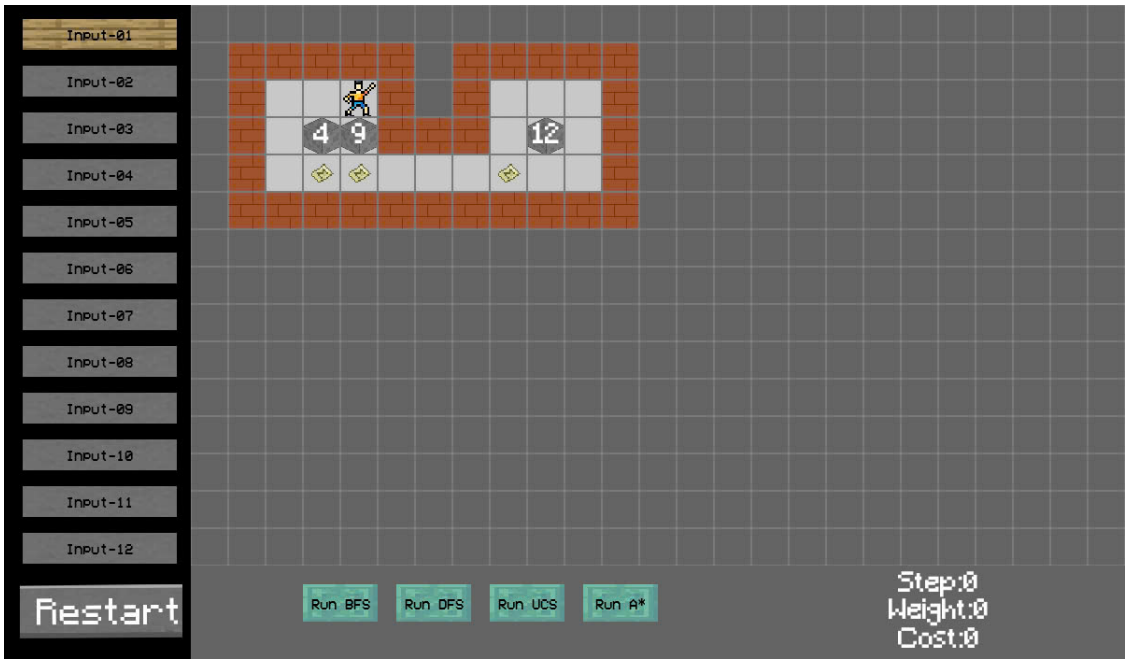
- **GUI**

  - Giao diện người dùng đẹp mắt với đầy đủ các yếu tố cần thiết.
  - Thể hiện trực quan các bước di chuyển của Ares để hoàn thành việc đẩy các đá giải quyết mê cung theo thuật toán được chọn.
  - Các thông số thay đổi theo thời gian thật: Steps, Weight, Cost.
  - Các phương thức để bắt đầu, dừng, bắt đầu lại chương trình cũng như lựa chọn màn chơi, lựa chọn thuật toán.

- **Lập trình**

  - Chương trình được cài đặt bằng ngôn ngữ lập trình Python với các thư viện cần thiết, các thuật toán tìm kiếm được xây dựng bởi các thành viên.

3 Giới thiệu các thành phần GUI



Giao diện bắt đầu chương trình

Nút chọn mê cung

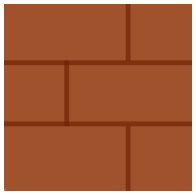
Nút chọn mê cung (khi chọn)

Nút chọn thuật toán

Nút chọn thuật toán (khi chọn)

Nút bắt đầu lại

Thông tin theo thời gian thực

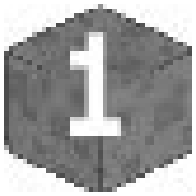


Ô tường

Ô mê cung

Người chơi (Ares)

Đích



Đá

Đá (khi ở đích)

4 Giới thiệu các thuật toán

4.1 Thuật toán tìm kiếm theo chiều rộng (Breath-first search)

BFS duyệt và tìm kiếm theo chiều rộng, bắt đầu từ một đỉnh gốc và duyệt các đỉnh kề trước khi tiếp tục đến các đỉnh xa hơn. Thuật toán sử dụng cấu trúc dữ liệu hàng đợi để lưu trữ các đỉnh, đồng thời đánh dấu lại các đỉnh đã duyệt để tránh duyệt lại.

Mã giả

Algorithm 1 Hàm BFS

```
Khởi tạo vị trí người chơi ban đầu, vị trí các đá ban đầu, vị trí các đích
trạng thái ban đầu = (vị trí người chơi ban đầu, vị trí các đá ban đầu)
Tạo hàng đợi và thêm (trạng thái ban đầu, 0, "")
Tạo danh sách thăm và thêm trạng thái ban đầu
số node = 0
while hàng đợi không rỗng do
    Lấy ((vị trí người chơi, vị trí các đá), số bước, đường đi) từ phần tử đầu tiên của hàng đợi, sau đó xóa trong hàng đợi
    if vị trí các đá trùng vị trí các đích then
        return dường đi, số node
    for hướng trong các hướng (lên, xuống, trái, phải) do
        vị trí người chơi mới = di chuyển(vị trí người chơi, hướng)
        if vị trí người chơi mới hợp lệ then
            if vị trí người chơi mới trùng vị trí các đá then
                vị trí đá mới = di chuyển(vị trí người chơi mới, hướng)
                if vị trí đá mới hợp lệ then
                    Cập nhật vị trí các đá mới
                    trạng thái mới = (vị trí người chơi mới, vị trí các đá mới)
                    if trạng thái mới chưa được thăm then
                        Thêm trạng thái mới vào danh sách thăm
                        Thêm (trạng thái mới, số bước + 1, đường đi + kí tự hướng viết hoa) vào hàng đợi
            else
                trạng thái mới = (vị trí người chơi mới, vị trí các đá)
                if trạng thái mới chưa được thăm then
                    Thêm trạng thái mới vào danh sách thăm
                    Thêm (trạng thái mới, số bước + 1, đường đi + kí tự hướng viết thường) vào hàng đợi
            số node += 1
return "", 0
```

4.2 Thuật toán tìm kiếm theo chiều sâu (Depth-first search)

DFS duyệt và tìm kiếm theo chiều sâu, bắt đầu từ một đỉnh gốc và đi sâu vào một nhánh cho đến khi không thể tiếp tục, sau đó quay lui và tiếp tục với nhánh khác. Thuật toán sử dụng cấu trú dữ liệu ngăn xếp để lưu trữ các đỉnh, đồng thời đánh dấu lại các đỉnh đã duyệt để tránh duyệt lại.

Mã giả

Algorithm 2 Hàm DFS

```
Khởi tạo vị trí người chơi ban đầu, vị trí các đá ban đầu, vị trí các đích
trạng thái ban đầu = (vị trí người chơi ban đầu, vị trí các đá ban đầu)
Tạo ngăn xếp và thêm (trạng thái ban đầu, 0, "")
Tạo danh sách thăm và thêm trạng thái ban đầu
số node = 0
while ngăn xếp không rỗng do
    Lấy ((vị trí người chơi, vị trí các đá), số bước, đường đi) từ phần tử cuối cùng của ngăn xếp, sau đó xóa trong ngăn xếp
    if vị trí các đá trùng vị trí các đích then
        return đường đi, số node
    for hướng trong các hướng (lên, xuống, trái, phải) do
        vị trí người chơi mới = di chuyển(vị trí người chơi, hướng)
        if vị trí người chơi mới hợp lệ then
            if vị trí người chơi mới trùng vị trí các đá then
                vị trí đá mới = di chuyển(vị trí người chơi mới, hướng)
                if vị trí đá mới hợp lệ then
                    Cập nhật vị trí các đá mới
                    trạng thái mới = (vị trí người chơi mới, vị trí các đá mới)
                    if trạng thái mới chưa được thăm then
                        Thêm trạng thái mới vào danh sách thăm
                        Thêm (trạng thái mới, số bước + 1, đường đi + kí tự hướng viết hoa) vào ngăn xếp
            else
                trạng thái mới = (vị trí người chơi mới, vị trí các đá)
                if trạng thái mới chưa được thăm then
                    Thêm trạng thái mới vào danh sách thăm
                    Thêm (trạng thái mới, số bước + 1, đường đi + kí tự hướng viết thường) vào ngăn xếp
            số node += 1
return "", 0
```

4.3 Thuật toán tìm kiếm chi phí đều (Uniform Cost Search)

UCS duyệt và tìm kiếm đường đi tối ưu dựa trên chi phí, ý tưởng giống với BFS nhưng ưu tiên mở rộng các đỉnh có chi phí nhỏ nhất từ đỉnh gốc đến đỉnh đó. Thuật toán sử dụng cấu trúc dữ liệu hàng đợi ưu tiên để lưu trữ các đỉnh, đồng thời đánh dấu lại các đỉnh đã duyệt để tránh duyệt lại.

Mã giả

Algorithm 3 Hàm UCS

```
Khởi tạo hàng đợi ưu tiên với (0, vị trí người chơi ban đầu, vị trí các đá ban đầu, "", 0)
Khởi tạo danh sách thăm với (vị trí người chơi ban đầu, vị trí các đá ban đầu)
số node = 0
while hàng đợi không rỗng do
    Lấy (chi phí hiện tại, vị trí người chơi, vị trí các đá, đường đi, khối lượng) từ phần tử đầu tiên của hàng đợi, sau đó xóa trong hàng đợi
    if vị trí các đá trùng vị trí các đích then
        return đường đi, số node, chi phí hiện tại, khối lượng
    for hướng trong các hướng (lên, xuống, trái, phải) do
        vị trí người chơi mới = di chuyển(vị trí người chơi, hướng)
        if vị trí người chơi mới không hợp lệ then
            Tiếp tục hướng tiếp theo
        vị trí các đá mới = sao chép(vị trí các đá)
        đẩy = False
        chi phí di chuyển = 1
        if vị trí người chơi mới trùng vị trí các đá then
            số thứ tự = số thứ tự của đá trong vị trí các đá mà vị trí người chơi mới trùng
            vị trí đá mới = di chuyển(vị trí người chơi mới, hướng)
            if vị trí đá mới không hợp lệ then
                Tiếp tục hướng tiếp theo
            đẩy = True
            khối lượng += khối lượng đá tại số thứ tự
            chi phí di chuyển += khối lượng đá tại số thứ tự
            Cập nhật vị trí các đá mới với vị trí đá tại số thứ tự = vị trí đá mới
        trạng thái mới = (vị trí người chơi mới, vị trí các đá mới)
        if trạng thái mới chưa được thăm then
            Thêm trạng thái mới vào danh sách thăm
            if đẩy == True then
                đường đi mới = đường đi + kí tự hướng viết hoa
            else
                đường đi mới = đường đi + kí tự hướng viết thường
            Thêm (chi phí hiện tại + chi phí di chuyển, vị trí người chơi mới, vị trí các đá mới, đường đi mới, khối lượng) vào hàng đợi
            số node += 1
return "", 0, 0, 0
```

4.4 Thuật toán tìm kiếm A*

A* duyệt và tìm kiếm đường đi tối ưu, kết hợp chi phí di chuyển thực tế và ước lượng chi phí đến đích, sử dụng một hàm đánh giá heuristic để ước lượng chi phí tối thiểu từ điểm hiện tại đến điểm đích. Thuật toán sử dụng cấu trúc

dữ liệu hàng đợi ưu tiên để lưu trữ các đỉnh, đồng thời đánh dấu lại các đỉnh đã duyệt để tránh duyệt lại.

Mã giả

Algorithm 4 Hàm A*

```
Khởi tạo hàng đợi ưu tiên với (0, vị trí người chơi ban đầu, vị trí các đá ban đầu, "", 0)
Khởi tạo danh sách thăm với (vị trí người chơi ban đầu, vị trí các đá ban đầu)
số node = 0
while hàng đợi không rỗng do
    Lấy (chi phí hiện tại, vị trí người chơi, vị trí các đá, đường dẫn, khối lượng) từ phần tử đầu tiên của hàng đợi, sau đó xóa trong hàng đợi
    if vị trí các đá trùng vị trí các đích then
        return đường đi, số node, chi phí hiện tại, khối lượng
    for hướng trong các hướng (lên, xuống, trái, phải) do
        vị trí người chơi mới = di chuyển(vị trí người chơi, hướng)
        if vị trí người chơi mới không hợp lệ then
            Tiếp tục hướng tiếp theo
        vị trí các đá mới = sao chép(vị trí các đá)
        đẩy = False
        chi phí di chuyển = 1
        if vị trí người chơi mới trùng vị trí các đá then
            số thứ tự = số thứ tự của đá trong vị trí các đá mà vị trí người chơi mới trùng
            vị trí đá mới = di chuyển(vị trí người chơi mới, hướng)
            if vị trí đá mới không hợp lệ then
                Tiếp tục hướng tiếp theo
            đẩy = True
            khối lượng += khối lượng đá tại số thứ tự
            chi phí di chuyển += khối lượng đá tại số thứ tự
            Cập nhật vị trí các đá mới với vị trí đá tại số thứ tự = vị trí đá mới
        trạng thái mới = (vị trí người chơi mới, vị trí các đá mới)
        if trạng thái mới chưa được thăm then
            Thêm trạng thái mới vào danh sách thăm
            chi phí heurestic = heuristic(vị trí các đá mới, vị trí các đích)
            if đẩy == True then
                đường đi mới = đường đi + kí tự hướng viết hoa
            else
                đường đi mới = đường đi + kí tự hướng viết thường
            Thêm (chi phí hiện tại + chi phí di chuyển + chi phí heurestic, vị trí người chơi mới, vị trí các đá mới, đường đi mới, khối lượng) vào hàng đợi
            số node += 1
return Không tìm thấy kết quả
```

4.5 Một số hàm hỗ trợ

Algorithm 5 Hàm kiểm tra vị trí hợp lệ

Lấy vị trí hoành độ, vị trí tung độ từ vị trí

return (0 <= vị trí hoành độ < chiều dài mê cung) và (0 <= vị trí tung độ < chiều cao mê cung) và (ô mê cung tại vị trí != ô tường)

Algorithm 6 Hàm di chuyển vị trí theo hướng

Lấy vị trí hoành độ, vị trí tung độ từ vị trí

return (vị trí hoành độ + lượng thay đổi hoành độ theo hướng, vị trí tung độ + lượng thay đổi tung độ theo hướng)

Algorithm 7 Hàm kiểm tra vị trí các đá trùng vị trí các đích

return Với mỗi vị trí đá trong vị trí các đá, kiểm tra vị trí đá có thuộc vị trí các đích

Algorithm 8 Hàm heuristic

return tổng(giá trị nhỏ nhất((khoảng cách Euclid từ vị trí đá đến vị trí đích +1) với mỗi vị trí đích trong vị trí các đích) với mỗi vị trí đá trong vị trí các đá)

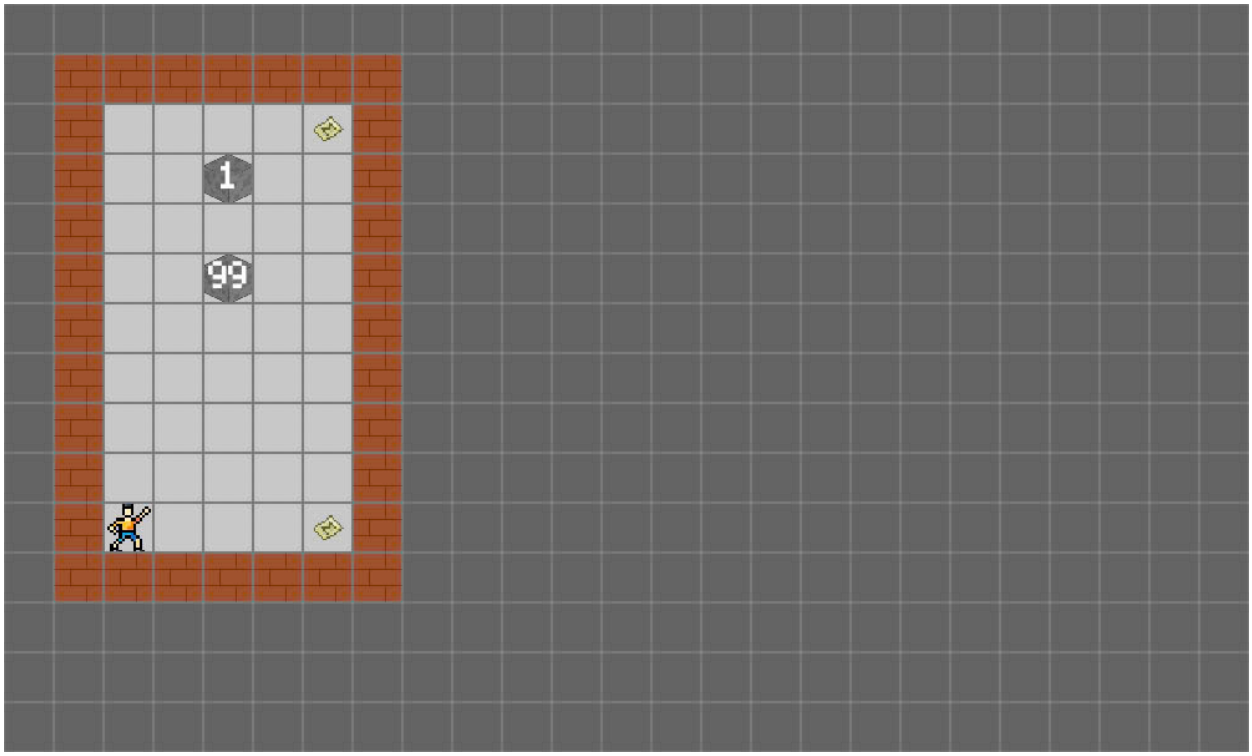
5 Giới thiệu các mê cung

5.1 Input-01

Không gian mê cung hình chữ nhật với không ô tường cản, hai đá với khối lượng chênh lệch nhiều là 1 và 99.

```
1 99
#######
#     .#
#  $  #
#     #
#  $  #
#     #
#     #
#     #
#     #
```

```
#@    .#
#######
```



Kết quả output-01

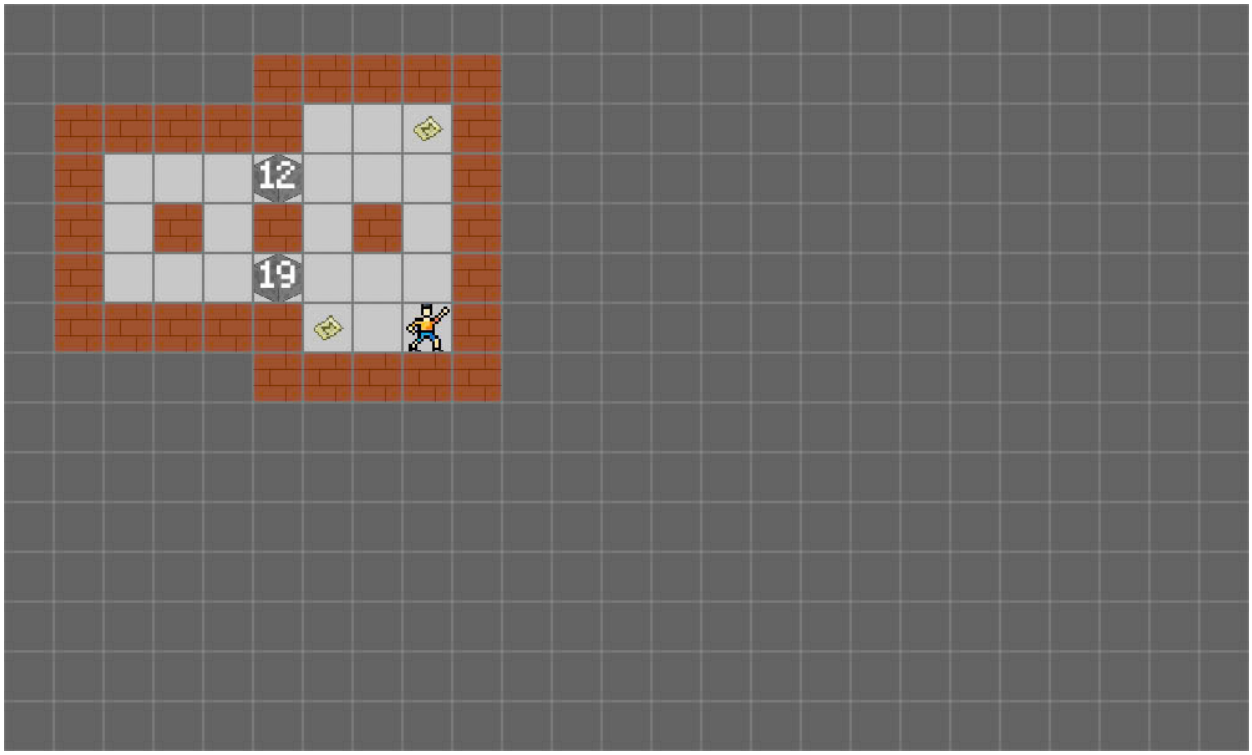
```
BFS
Steps: 23, Weight: 696, Node: 99138, Time (ms): 1049.28,
  ↪ Memory (MB): 6.92
uuuuurRuUluRRddDDDDDldR
DFS
Steps: 347, Weight: 2108, Node: 3960, Time (ms): 34.34, Memory
  ↪ (MB): 0.28
rrrruullllluurrrruullllluurrDrrddlLrrddllllluuRRRllddrrrruUllllluur
rrrDDllllluurrDrrdDllllddrrrrUllllluurRlldrrrrUllllluurrrrDDllllluu
rrrDrdLrDllllddrrrrUllllluurRlldrrrrUllllluurrrDllldrrrruLrddllll
luururDrrddllllluuururrrDllllddrRlldrrrruuLrddlluUrrddllllluuurRl
luurrrrdDLrDllllddrrrrUllllluurRlldrrruUllluurrrDDrDLrDllllluurrD
rruullllddrRdrUluRdrUUUUUU
UCS
```

```
Steps: 31, Weight: 504, Nodes: 72620, Time (ms): 9810.45,
  ↪ Memory (MB): 14.28
uuuuuuurRRurDDDDDDDuuullUUUluRR
A*
Steps: 31, Weight: 504, Nodes: 65549, Time (ms): 7620.55,
  ↪ Memory (MB): 13.18
uuuuurRRdrUUUldlluurDDDDDDDldRR
```

5.2 Input-02

Không gian mê cung hẹp với một số ô tường cản, hai đá với khối lượng không chênh lệch nhiều là 12 và 19.

```
12 19
%%%#####
#####  .#
#   $   #
# # # # #
#   $   #
#####. @#
%%%#####
```



Kết quả output-02

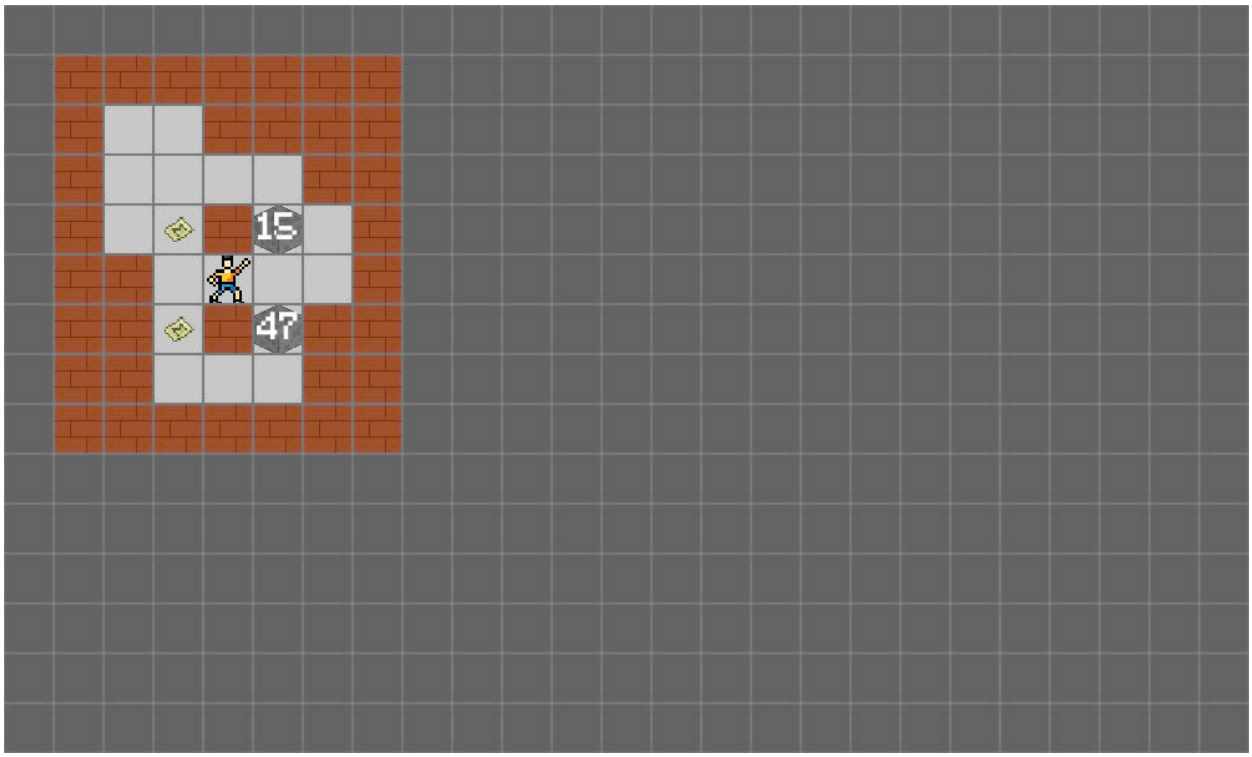
```
BFS
Steps: 27, Weight: 134, Node: 5389, Time (ms): 74.63, Memory
 ↪ (MB): 0.32
uuullLLddRllluuRRRRRldDrruU
DFS
Steps: 99, Weight: 338, Node: 2211, Time (ms): 34.10, Memory
 ↪ (MB): 0.04
lluLLuuRRRlllllddRRRRllllluurrrrrrurDlllllldrrrrdrruLrUdlLrruUll
llllddrrRRllllluurrrrrrddLdlUrruullDD
UCS
Steps: 27, Weight: 134, Nodes: 3039, Time (ms): 381.48, Memory
 ↪ (MB): 0.64
uuullLLddRllluuRRRRRldDrruU
A*
Steps: 27, Weight: 134, Nodes: 1953, Time (ms): 281.36, Memory
 ↪ (MB): 0.48
```

```
uuullLLddRllluuRRRRRldDrruU
```

5.3 Input-03

Không gian mê cung hẹp với một số ô tường cản, hai đá với khối lượng chênh lệch tương đối là 15 và 47.

```
15 47
#######
#  ####
#    ##
# .#$ #
## @  #
##.#$##
##   ##
#######
```



Kết quả output-03

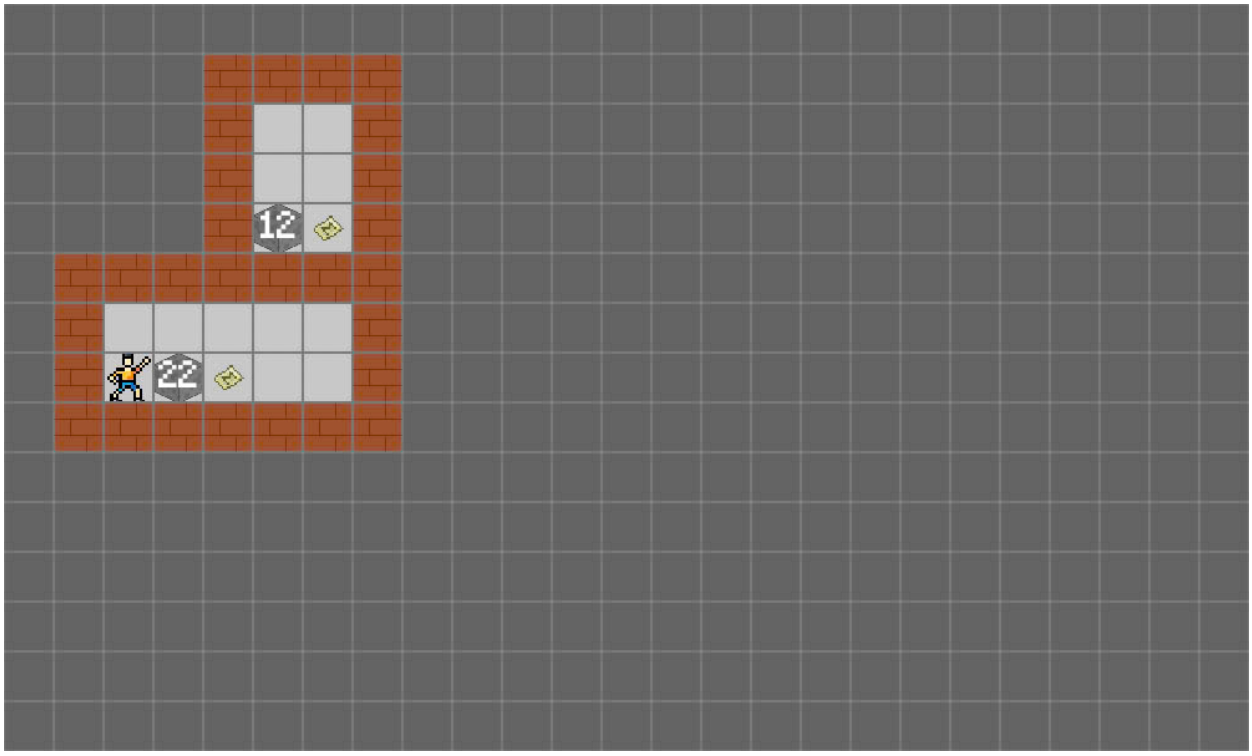
```
BFS
Steps: 91, Weight: 620, Node: 6001, Time (ms): 106.37, Memory
 ↪ (MB): 0.31
luurrDrdLuulldddrrUULrddlluUUluurDrrDrdLddlluuUluRdddrruuLrdd
lluUUluurDDDrruuLrddlluluurD
DFS
Steps: 91, Weight: 620, Node: 4645, Time (ms): 70.13, Memory
 ↪ (MB): 0.25
luurrDrdLuulldddrrUULrddlluUUluurDrrDrdLddlluuUluRdddrruuLrdd
lluUUluurDDDrruuLrddlluluurD
UCS
Steps: 91, Weight: 620, Nodes: 2428, Time (ms): 218.09, Memory
 ↪ (MB): 0.53
luurrDrdLuulldddrrUULrddlluUUluurDrrDrdLddlluuUluRdddrruuLrdd
lluUUluurDDDrruuLrddlluluurD
A*
Steps: 91, Weight: 620, Nodes: 2447, Time (ms): 316.98, Memory
 ↪ (MB): 0.53
luurrDrdLuulldddrrUULrddlluUUluurDrrDrdLddlluuUluRdddrruuLrdd
lluUUluurDDDrruuLrddlluluurD
```

5.4 Input-04

Không gian mê cung không thể giải với hai khu vực bị ngăn cách, trong đó có thể hoàn thành đẩy đá có khối lượng là 22 ở một khu vực, không thể di chuyển tới khu vực còn lại để hoàn thành đẩy đá có khối lượng là 12.

```
12 22
   ####
   #  #
   #  #
   #$.#
```

```
#######
#     #
#@$.  #
#######
```



Kết quả output-04

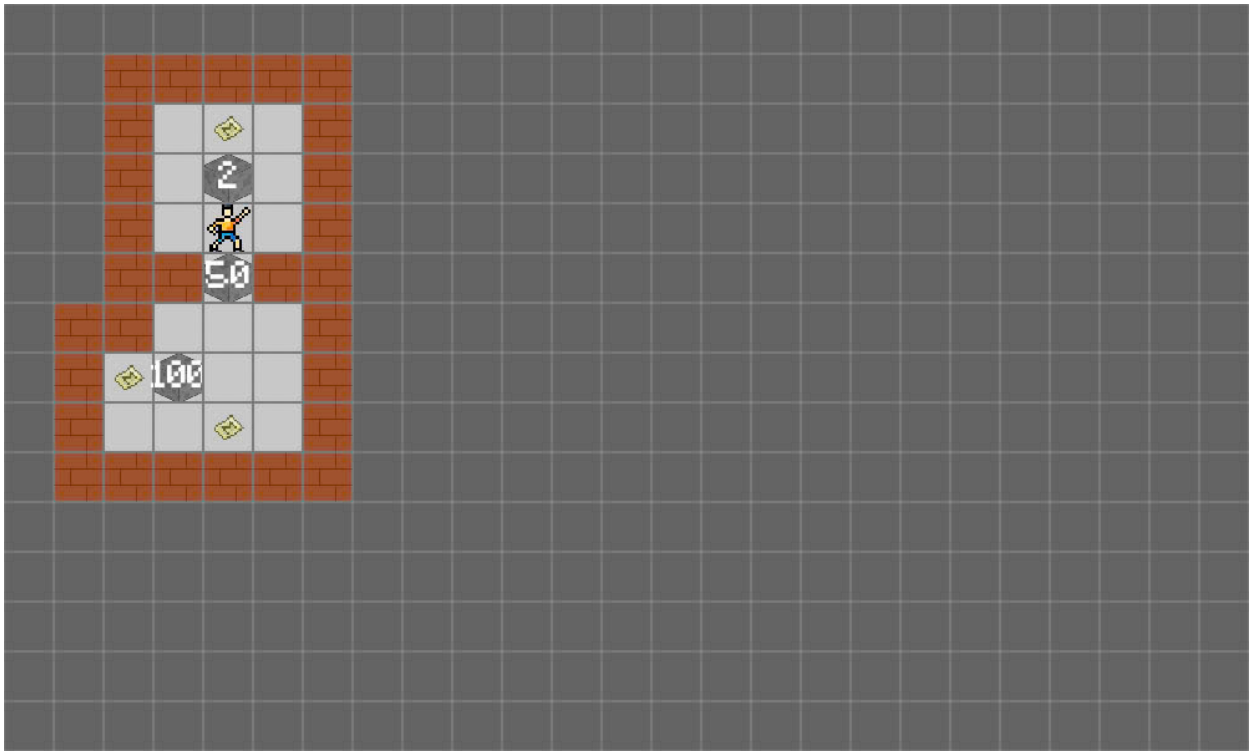
```
BFS
Steps: 0, Weight: 0, Node: 0, Time (ms): 0.98, Memory (MB):
  ↪ 0.01

DFS
Steps: 0, Weight: 0, Node: 0, Time (ms): 1.03, Memory (MB):
  ↪ 0.00

UCS
Steps: 0, Weight: 0, Nodes: 0, Time (ms): 3.03, Memory (MB):
  ↪ 0.01
```

```
A*
Steps: 0, Weight: 0, Nodes: 0, Time (ms): 3.00, Memory (MB):
 ↪ 0.01
```

5.5 Input-05

Không gian mê cung với ba đá có khối lượng chênh lệch nhiều là 2, 50 và 100, đường thoát khu vực kín bị chặn bởi đá có khối lượng là 50.

```
2 50 100
%#####
%# . #
%# $ #
%# @ #
%##$##
##   #
#.$  #
#  . #
######
```



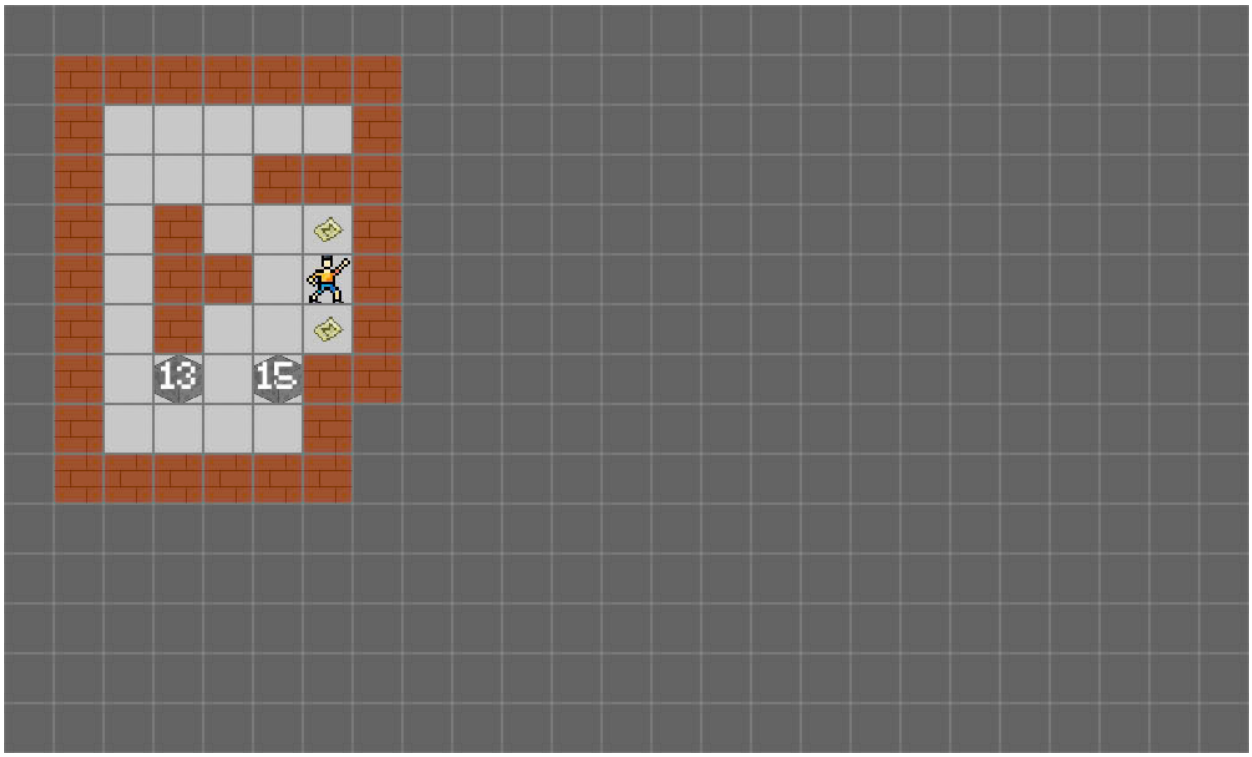
Kết quả output-05

```
BFS
Steps: 6, Weight: 252, Node: 258, Time (ms): 2.02, Memory
  ↪ (MB): 0.02
UdDDDL
DFS
Steps: 19, Weight: 402, Node: 86266, Time (ms): 1164.98,
  ↪ Memory (MB): 5.58
DDlDldRuurrdLLruuuU
UCS
Steps: 6, Weight: 252, Nodes: 1718, Time (ms): 212.70, Memory
  ↪ (MB): 0.37
UdDDDL
A*
Steps: 6, Weight: 252, Nodes: 847, Time (ms): 102.65, Memory
  ↪ (MB): 0.15
UdDDDL
```

5.6 Input-06

Không gian mê cung với một số đường ngách cụt khi đẩy đá vào, cần thực hiện đường đi khéo léo để hoàn thành đẩy hai đá có khối lượng không chênh lệch nhiều là 13 và 15.

```
13 15
#######
#     #
#   ###
# #  .#
# ## @#
# #  .#
# $ $##
#    #%
######%
```



Kết quả output-06

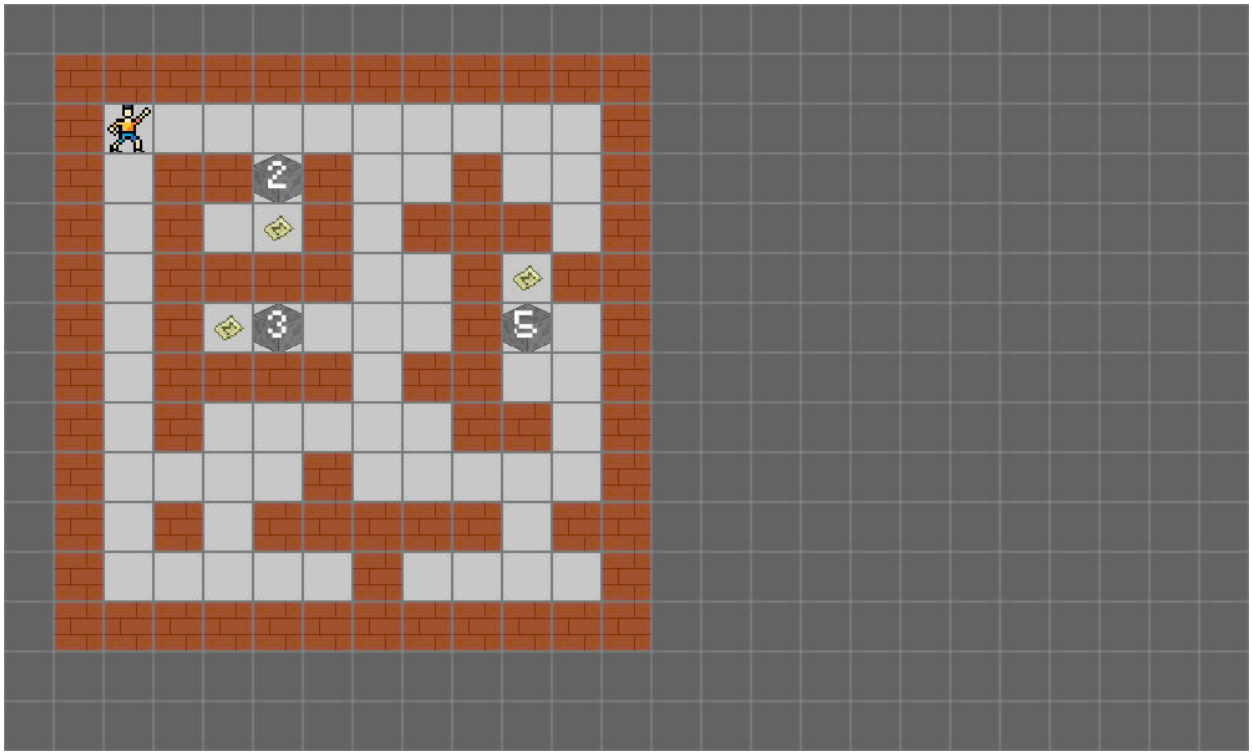
```
BFS
Steps: 36, Weight: 112, Node: 8411, Time (ms): 126.94, Memory
 ↪ (MB): 0.40
dllddrUUUdddllluRRdrUluRdllluuuurrdR
DFS
Steps: 36, Weight: 108, Node: 4411, Time (ms): 66.55, Memory
 ↪ (MB): 0.21
ldlddrUldlluRRuRlddrUUUdldlluuuurrdR
UCS
Steps: 36, Weight: 108, Nodes: 3220, Time (ms): 364.65, Memory
 ↪ (MB): 0.66
ldlddrUluRlddlluRRdrUUUdldlluuuurrdR
A*
Steps: 36, Weight: 108, Nodes: 2660, Time (ms): 354.06, Memory
 ↪ (MB): 0.70
ldlddrUluRlddlluRRdrUUUdldlluuuurrdR
```

5.7 Input-07

Không gian mê cung lớn với nhiều đường ngách rẽ nhánh cụt, ba đá với khối lượng là 2, 3 và 5 chỉ có thể được hoàn thành đẩy với mỗi đích tương ứng.

```
2 3 5
############
#@         #
# ##$#  #  #
# # .# ### #
# ####  #.##
# #.$   #$ #
# #### ##  #
# #     ## #
#    #     #
```

```
# # ##### ##
#     #    #
############
```



Kết quả output-07

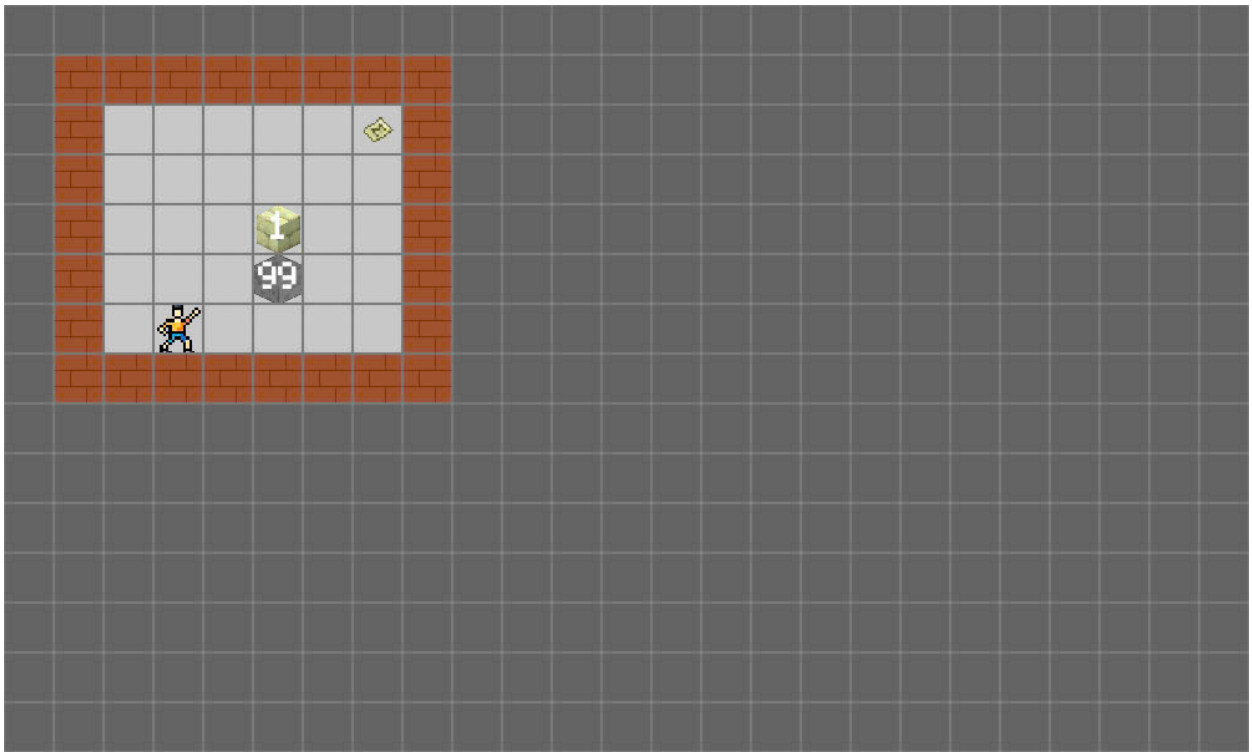
```
BFS
Steps: 26, Weight: 10, Node: 570, Time (ms): 11.82, Memory
 ↪ (MB): 0.01
rrrDurrddddlLrrdddrrrruulU
DFS
Steps: 50, Weight: 10, Node: 205, Time (ms): 2.99, Memory
 ↪ (MB): 0.01
rrrrrddddlLrrddrdrrruulUrdddllllullldlluuuuuuurrrD
UCS
Steps: 26, Weight: 10, Nodes: 277, Time (ms): 43.61, Memory
 ↪ (MB): 0.06
rrrDurrddddlLrrddrdrrruulU
A*
```

```
Steps: 26, Weight: 10, Nodes: 245, Time (ms): 44.16, Memory
  ↪ (MB): 0.04
rrrDurrdddddlLrrddrdrrruulU
```

5.8 Input-08

Không gian mê cung hình chữ nhật với không ô tường cản, hai đá với khối lượng chênh lệch lớn là 1 và 99, trong đó đá có khối lượng là 1 đã ở một đích, đá có khối lượng là 99 ở xa đích còn lại hơn là đích mà đá có khối lượng là 1 đã chiếm.

```
1 99
########
#      .#
#       #
#   *   #
#   $   #
# @     #
########
```



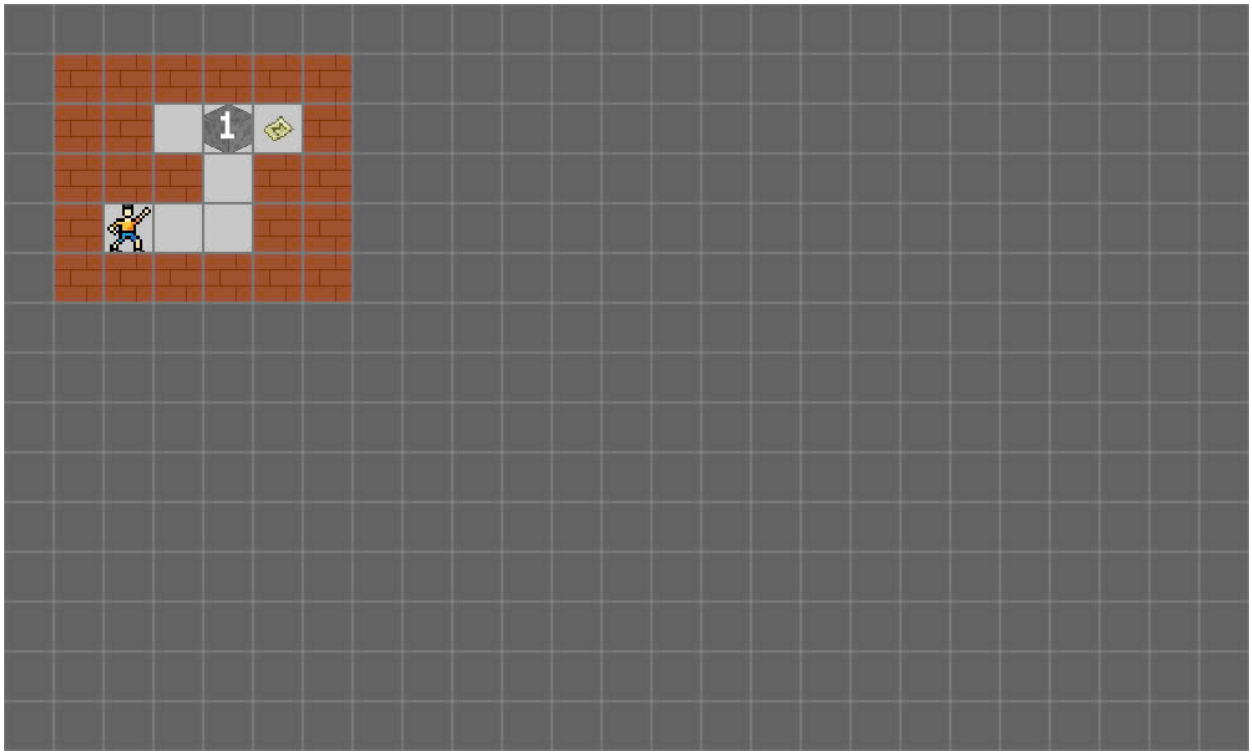
Kết quả output-08

```
BFS
Steps: 9, Weight: 495, Node: 1972, Time (ms): 13.01, Memory
 ↪ (MB): 0.08
urRRdrUUU
DFS
Steps: 51, Weight: 105, Node: 170, Time (ms): 0.99, Memory
 ↪ (MB): 0.01
rrrruulLrrddllllluurRRRlllddrrrrruUllllddrrrUrruU
UCS
Steps: 13, Weight: 103, Nodes: 2767, Time (ms): 169.11, Memory
 ↪ (MB): 0.67
uurRRddlUrrUU
A*
Steps: 13, Weight: 103, Nodes: 2423, Time (ms): 182.70, Memory
 ↪ (MB): 0.58
uurRRddlUrrUU
```

5.9 Input-09

Không gian mê cung không thể giải do vị trí bắt đầu của người chơi không nằm vào ô duy nhất có thể hoàn thành đẩy đá có khối lượng là 1 vào đích.

```
1
######
## $.#
### ##
#@  ##
######
```



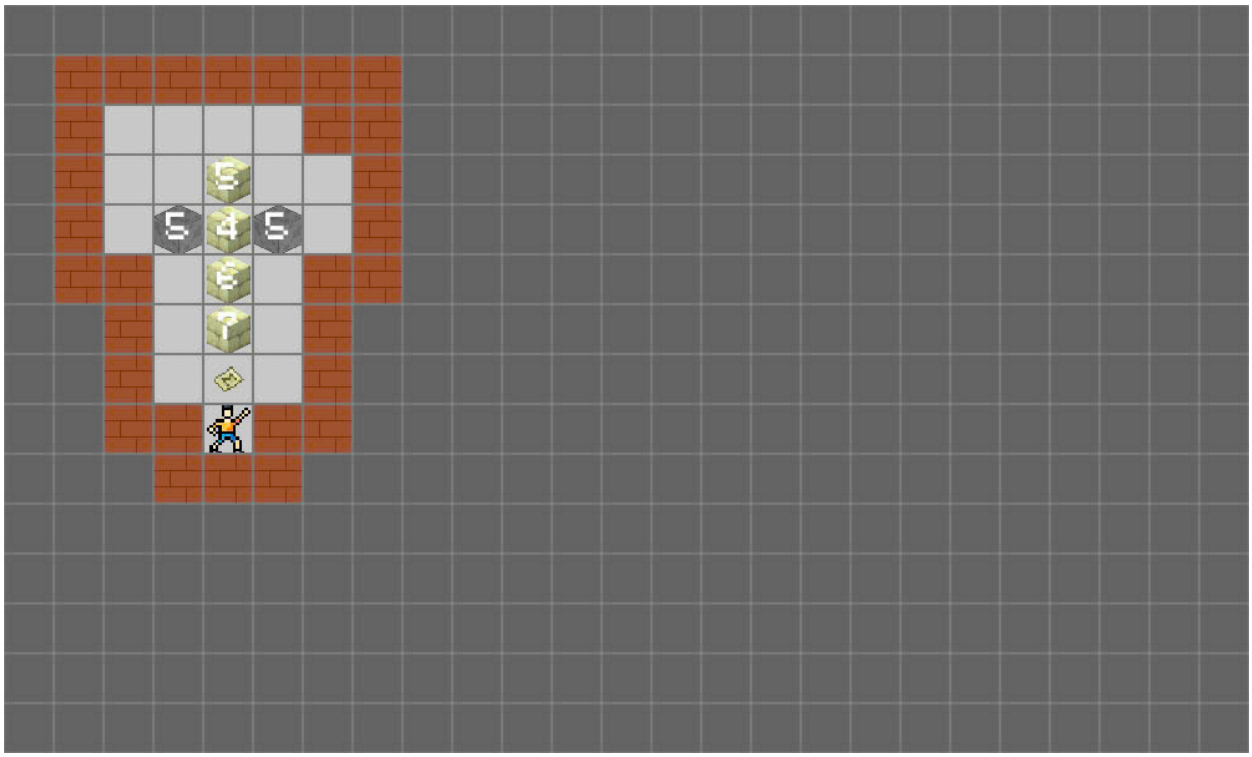
Kết quả output-09

```
BFS
Steps: 0, Weight: 0, Node: 0, Time (ms): 0.00, Memory (MB):
  ↪ 0.00

DFS
Steps: 0, Weight: 0, Node: 0, Time (ms): 1.36, Memory (MB):
  ↪ 0.00

UCS
Steps: 0, Weight: 0, Nodes: 0, Time (ms): 0.00, Memory (MB):
  ↪ 0.00

A*
Steps: 0, Weight: 0, Nodes: 0, Time (ms): 1.00, Memory (MB):
  ↪ 0.00
```

5.10 Input-10

Không gian mê cung với không ô tường cản, sáu đá với khối lượng không chênh lệch nhiều là 5, 5, 4, 5, 6 và 7, trong đó có bốn đá với khối lượng là 5, 4, 6, 7 đã ở đích, người chơi đang ở vị trí của một trong hai đích còn lại.

```
5 5 4 5 6 7
#######
#     ##
#      #
#      #
##$*$##
%# * #%
%# . #%
%##+##%
%%###%%
```



Kết quả output-10

BFS

Steps: 46, Weight: 108, Node: 11606540, Time (ms): 236005.06,
↪ Memory (MB): 650.45

uluUUrrDLDDrUUUlllluRurDDDDlUUluRurDDDurruLulDD

DFS

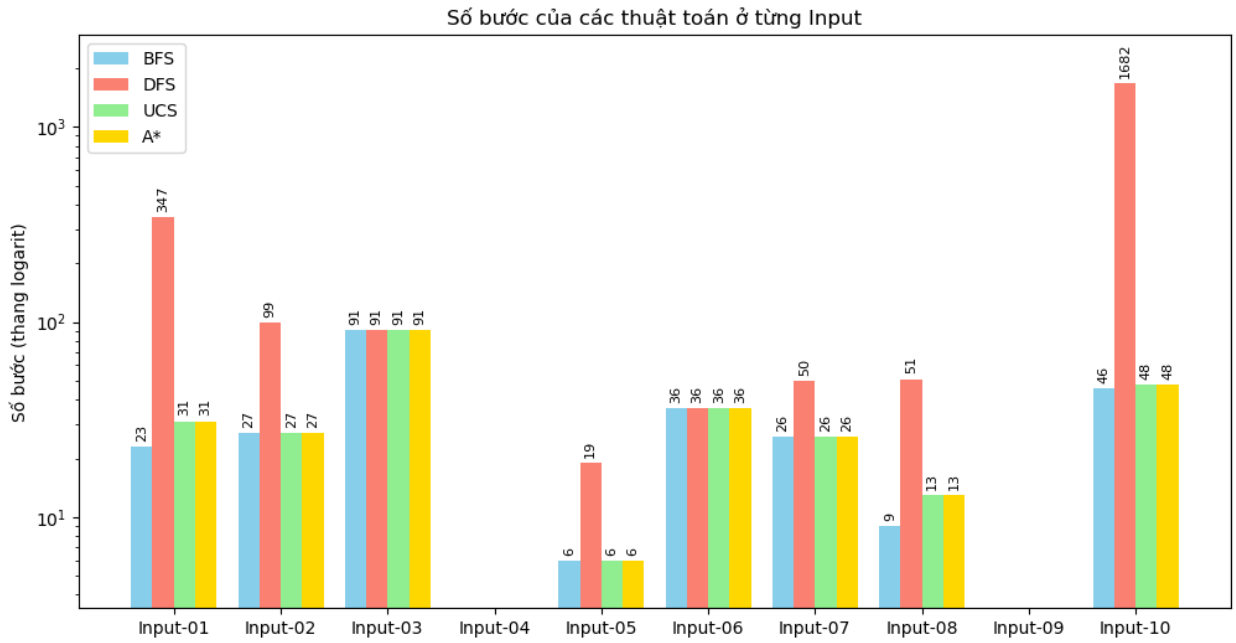
Steps: 1682, Weight: 1843, Node: 5694757, Time (ms):
↪ 104806.12, Memory (MB): 282.32

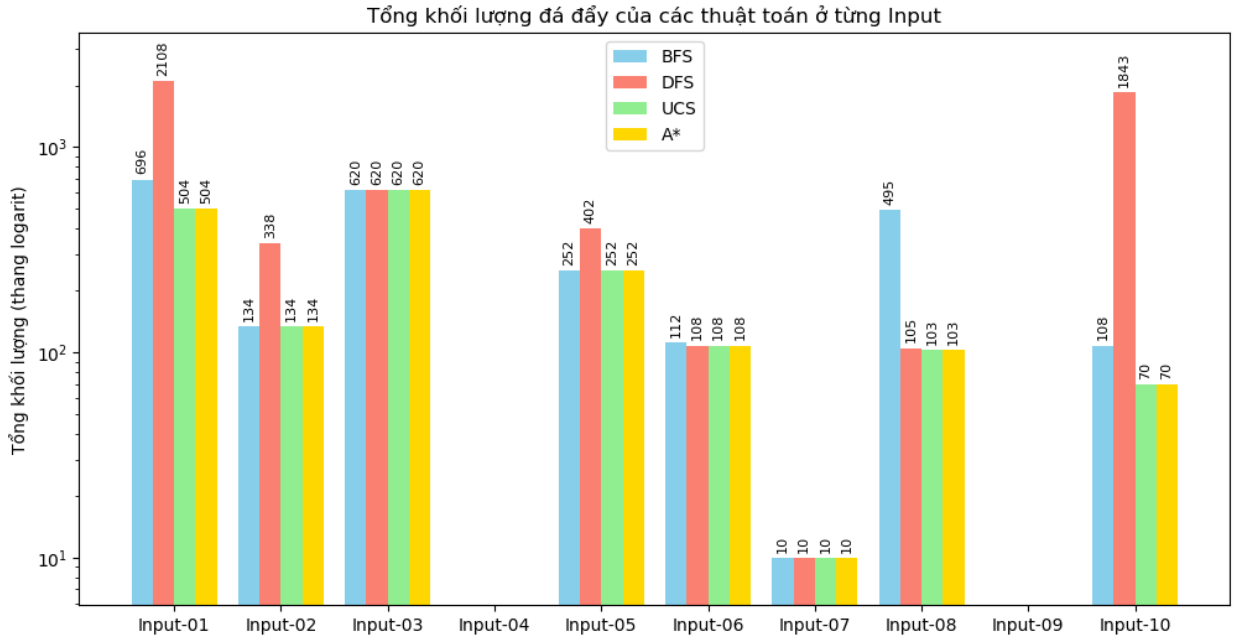
uruUddlluRlUdrdrUlluUrDlddrruLdlUruullluRdrddlUUluurrrdrdLLrDuuu
llddRRlluurrrdrdLuullDRlDluurrrDlldDrrUruLLrdLruullldRRlldRluu
rrrDlDlluurrrdrdLLDruullldRRlluurrrdrdLdlDDrUldlUrrUldlUrrUruLL
rddlUruullldRRlluurrrdrdLuullDRlDluurrrDlldDrrddllUrruUruLLrdL
ruullldRRurDrdLdlUruulDDruullldRRlluurrrdrdLddllURlddrruUlUdrU
ldlUluurrrDrdLLruullldRRlluurrDrDLrDlldrrUUluuulldRldRluurrrd
DLrDlldrrUlluUrrDlldrrUUllluluurrrdrdLLruullldRluurDRlDluurrr
DlldDrUlluurrrdrdLLrddldlUrruuuLulldRRlldRluurrrDlldDrrddllUrru
UruLLrddldlUrruulDrddlluRlddrruUluurDDluuulldRRlldRluurrrDllddr
rUruLLrddllddrrUlluuRlluRurrdrdLLrDlldrrUlluUrrDlldrrUUlluluu
rrrdrdLLruullldRluurDRDrruLdlDrdLruuuLulldRRlldRluurrrDlDlluur
rrdrdLLrddldlUrruulDrddlluRdrUluurDululldRRlluurrrdrdLdldlUrruu
lDrdLruullldRRlluurrrdrdLdldlUrruulDruullldRRlddrdrUlluurDlddr
ruLdlUruulluurrrdrdLulldDrdrUllulluurrrdDLrDllUrruullldRRlddrrur
uLdldddrUldlUrrUldlUrrUldlUluurrDDrruLdlllluurrDrddLrddlluUrruuu
llldRRlDluurrrDlldDrrddllUrruUruLLDrddllURlddrruUluurDDluuulldR
RlldRluurrrDlldrrUruLLrddllddrrUlluuRlluRurrdrdLLrDlldrrUlluU
rrDlldrrUUllulluurrDrDLrDlldrrUUluurrdLdlUruullldRRlluurrrdrd
LuullDRlDluurrrDlldDrrddllUrruUruLLrdLruullldRRurDrdLdlUruulDDr
uullldRluurrrddddllURlddrruUUdllUluRurrDDLrDllUluurrrdddlddrUl
luuluurrDDrDLruulullldRRdrruLddlddrUluuruLdlluRRldDrddlUUrurDllU
rrruLulDDrdLruulullldRRddd1UrruruLullldRdrddrUlluuRlluurrrdLdll
uurDrrrdLullDluurrdDDDrUldlUrrUruLLullldRRlldRluurrrDlldrrUruLL
rddlluRlluRdrdrUlllluurrrDDlllluurrDrdLruullldRdrdrUlluluurrdDll
uurrrdrdLLrdLruullldRRlluurrrdrdLdldlUrruulDrdLruullldRluurrrdd

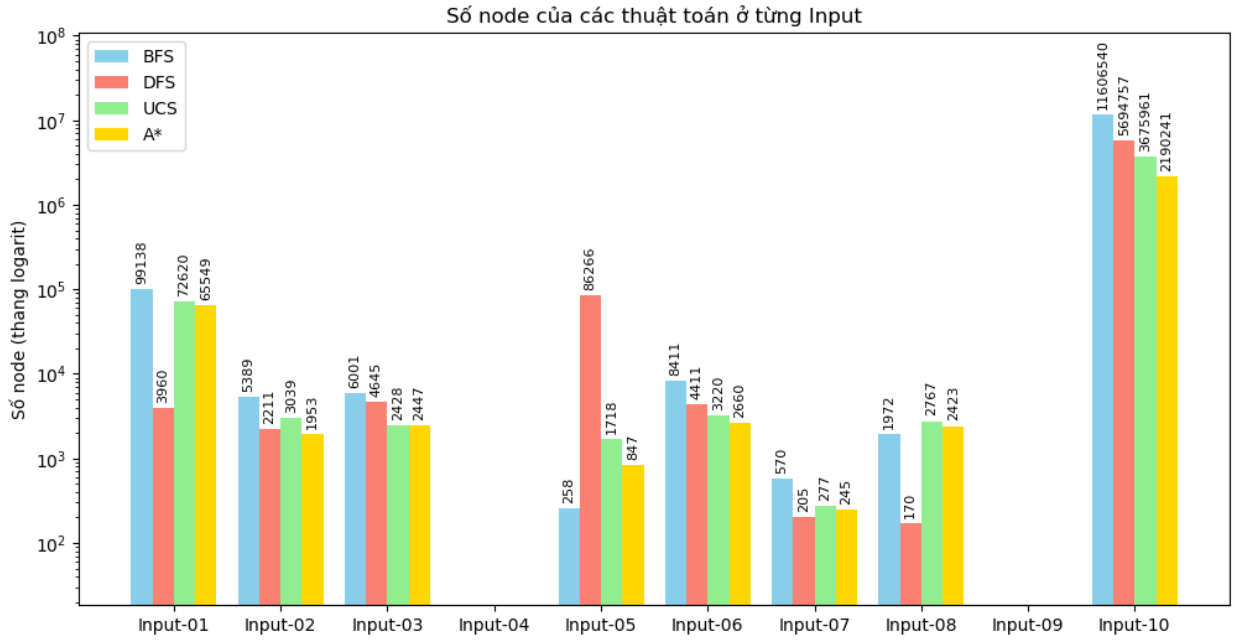
```
dldlUrUdlUrruulDrdLruullldRRlDrrruLLdlluRurDDrddllUUluRRldddrru
uruLdlDruuLrddlluluRRurDrdLLruullldRluurrdD
UCS
Steps: 48, Weight: 70, Nodes: 3675961, Time (ms): 442077.00,
  ↪ Memory (MB): 842.61
uruUddlluURDDuuuurrdLulDDDrUluurrdLulDDuulldRurD
A*
Steps: 48, Weight: 70, Nodes: 2190241, Time (ms): 264685.96,
  ↪ Memory (MB): 533.97
uruUddlluURDDuuuurrdLulDDDrUluulldRurDDuurrdLulD
```

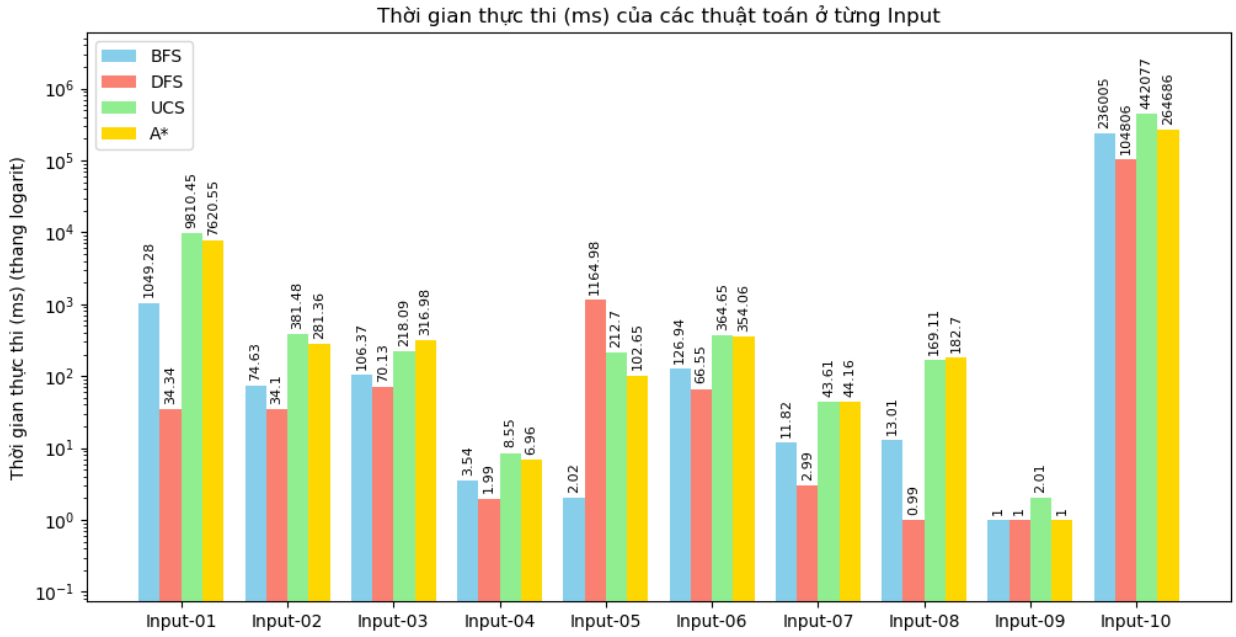
---

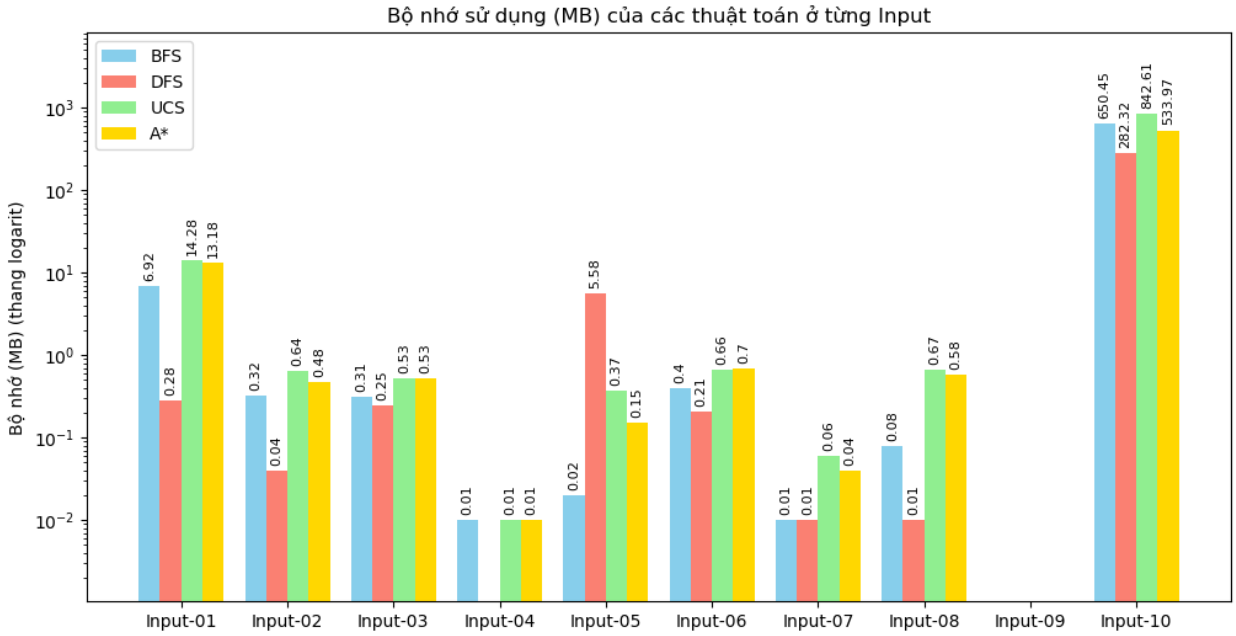
6 So sánh mức độ hiệu quả các thuật toán




Tổng khối lượng đá đẩy của các thuật toán ở từng Input
BFS
DFS
UCS
A*
Tổng khối lượng (thang logarit)
10^3
10^2
10^1
696
2108
504
504
134
338
134
134
620
620
620
620
252
402
252
252
112
108
108
108
10
10
10
10
495
105
103
103
108
1843
70
70
Input-01
Input-02
Input-03
Input-04
Input-05
Input-06
Input-07
Input-08
Input-09
Input-10


Số node của các thuật toán ở từng Input
BFS
DFS
UCS
A*
Số node (thang logarit)
10^8
10^7
10^6
10^5
10^4
10^3
10^2
99138
3960
72620
65549
5389
2211
3039
1953
6001
4645
2428
2447
258
86266
1718
847
8411
4411
3220
2660
570
205
277
245
1972
170
2767
2423
11606540
5694757
3675961
2190241
Input-01
Input-02
Input-03
Input-04
Input-05
Input-06
Input-07
Input-08
Input-09
Input-10





Nhận xét:

Theo các Input:

- **Input-01**: DFS có số bước đi nhiều nhất và khối lượng đá đẩy lớn nhất, nhưng duyệt ít node nhất, thời gian thực thi nhanh nhất và sử dụng ít

bộ nhớ nhất.

- **Input-02**: DFS có số bước đi nhiều nhất và khối lượng đá đẩy lớn nhất, nhưng duyệt ít node, thời gian thực thi nhanh nhất và sử dụng ít bộ nhớ nhất.
- **Input-03**: Các thuật toán đều có số bước đi, khối lượng đá đẩy giống nhau. UCS và A* đều duyệt ít node hơn nhưng thời gian thực thi và sử dụng bộ nhớ đều tối ưu kém hơn hai thuật toán còn lại.
- **Input-04**: Mê cung không giải được nên không có thông số về số bước đi, khối lượng đá đẩy và số node. DFS có thời gian thực thi ngắn nhất và sử dụng bộ nhớ ít nhất để tìm ra kết luận không thể giải.
- **Input-05**: DFS có thông số tối ưu kém nhất ở tất cả các tiêu chí, trong khi BFS có thông số tối ưu tốt nhất ở tất cả các tiêu chí.
- **Input-06**: Các thuật toán đều có số bước đi giống nhau nhưng BFS có khối lượng đá đẩy nhỉnh hơn một chút. UCS và A* đều duyệt ít node hơn nhưng thời gian thực thi và sử dụng bộ nhớ đều tối ưu kém hơn hai thuật toán còn lại.
- **Input-07**: DFS tuy có số bước đi nhiều nhất nhưng những thông số còn lại đều ở mức tối ưu tốt nhất. UCS và A* đều duyệt tương đối ít node nhưng thời gian thực thi và sử dụng bộ nhớ đều tối ưu kém hơn hai thuật toán còn lại.
- **Input-08**: BFS tìm được đường đi ngắn nhất nhưng UCS và A* lại có khối lượng đá đẩy ít nhất trong khi ở các thông số số node, thời gian thực thi, bộ nhớ sử dụng thì DFS vẫn có mức tối ưu tốt nhất.
- **Input-09**: Mê cung không giải được nên không có thông số về số bước đi, khối lượng đá đẩy và số node. UCS có thời gian thực thi để tìm ra kết luận không thể giải hơi kém hơn những thuật toán còn lại, bộ nhớ sử dụng ở cả bốn thuật toán đều ở mức rất thấp, gần như không có.
- **Input-10**: Mê cung khó giải nhất với DFS khi có số bước đi, khối lượng đá đẩy đều lớn hơn đáng kể so với những thuật toán khác. Cả bốn thuật

toán đều có thông số về số node, thời gian thực thi, bộ nhớ sử dụng cao hơn hẳn so với những Input khác.

Theo các tiêu chí:

- **Số bước**: Ở tất cả các Input, DFS đều có xu hướng có số bước đi nhiều nhất, do bản chất thuật toán tìm kiếm theo chiều sâu nên sẽ mất nhiều bước với những mê cung trống trải và ít vật cản, ngược lại thì sẽ có kết quả tốt hơn với những mê cung nhiều nhánh và ngõ cụt. Các thuật toán còn lại đều có kết quả giống nhau ở nhiều Input trừ ở Input-01 và Input-08 thì BFS có số bước đi ít hơn UCS và A* một chút.
- **Khối lượng**: Ở tất cả các Input, UCS và A* đều có kết quả tối ưu nhất do bản chất thuật toán ưu tiên tìm đường đi có chi phí tối ưu. DFS có kết quả kém tối ưu nhất đáng kể so với những thuật toán khác ở bốn Input, trừ Input-08 là trường hợp duy nhất BFS có kết quả tối ưu kém hơn hẳn những thuật toán khác do bản chất các thuật toán không xét đến việc tối ưu chi phí.
- **Số node**: DFS có xu hướng có số node ít và ở mức tối ưu tốt trong nhiều Input trừ Input-05 là trường hợp lại có kết quả kém hơn hẳn những thuật toán khác. BFS có kết quả thuộc mức tối ưu kém nhất ở nhiều Input với số node lớn nhất trừ ở Input-05 lại có kết quả tối ưu tốt nhất.
- **Thời gian**: DFS có thời gian thực thi ngắn nhất ở tất cả các trường hợp trừ Input-05, UCS và A* đều có kết quả tối ưu kém nhất với thời gian thực thi thuộc nhóm dài nhất ở tất cả các trường hợp trừ Input-05.
- **Bộ nhớ**: Mối quan hệ tương đối giữa các thuật toán trong tiêu chí bộ nhớ rất giống với mỗi quan hệ tương đối đó trong tiêu chí thời gian. DFS có kết quả tối ưu tốt nhất và UCS, A* có kết quả tối ưu kém nhất ở mọi trường hợp trừ Input-05

7 Video Demo chương trình

https://www.youtube.com/watch?v=j-rLDRqMP6w