CS106.M21.KHCL

DFS/BFS/UCS for Sokoban

19521638 – Trương Hưng Huy

Mục lục

[1. Cơ Chế Game 2](#_Toc98937992)

[2. Trạng Thái và Mô Hình Hoá 2](#_Toc98937993)

[3. Các Thuật Toán 4](#_Toc98937994)

[3.1 Depth First Search (DFS) 4](#_Toc98937995)

[3.2 Breadth First Search (BFS) 5](#_Toc98937996)

[3.3 Uninformed Cost Search (UCS) 6](#_Toc98937997)

[4. Kết Quả 7](#_Toc98937998)

[5. Kết Luận 9](#_Toc98937999)

# 1. Cơ Chế Game

Sokoban là game giải đố với bản đồ chứa một ma trận các hình vuông. Các hình vuông có thể là thùng hàng, kho hàng (đích đến), ô trống hoặc tường.

Trạng thái khởi đầu gồm vị trí của player tại điểm x, y nhất định trên ma trận, các thùng hàng và kho hàng. Player có thể di chuyển sang trái, phải, lên, xuống. Player có thể đẩy tối đa 1 thùng hàng vào mỗi ô trống trên bản đồ. Player không thể kéo thùng hàng. Số lượng các thùng hàng và kho hàng của mỗi bản đồ là bằng nhau, player sẽ thắng khi tất cả các vị trí của thùng hàng trùng với vị trí của kho hàng. Player thua khi thùng hàng bị kẹt vào góc mà vị trí bị kẹt không là vị trí kho hàng.

# 2. Trạng Thái và Mô Hình Hoá

Sokoban có thể được xem như là một bài toán tìm kiếm trong đó chúng ta sẽ tìm đường đi từ trạng thái khởi đầu đến trạng thái kết thúc.

Sokoban được mô hình hoá như sau:

* Các thùng hàng được quy ước là số ‘3’
* Các kho hàng được quy ước là số ‘4’
* Tường được quy ước là số ‘1’
* Player được quy ước là số ‘2’
* Những vị trí rỗng trên map được quy ước là số ‘0’
* Những vị trí thùng hàng đã được đẩy vào kho hàng được quy ước là số ‘5’

Dựa vào các quy ước trên, ta có thể mô hình hoá bản đồ bên dưới như sau:

Chart

Description automatically generated

Hình 1: Mô hình hoá Sokoban

* Không gian trạng thái (states) có dạng:

( (xp, yp), ( (xb1, yb1), (xb2, yb2), … , (xbn, ybn) ) ), trong đó:

* + xp, yp là toạ độ của player trên trục x, y. Ví dụ ở Hình 1, player có toạ độ

(xp, yp) = (5, 5)

* + xbi, ybi là toạ độ của thùng hàng thứ i trên map. Ví dụ ở Hình 1, 2 thùng hàng có toạ độ như sau: ( (xb1, yb1), (xb2, yb2) ) = ( (1, 3), (4, 4) ).
* Các hành động hợp lệ (actions) gồm Lên, Xuống, Trái, Phải, được mô hình hoá như sau:

|  |  |  |
| --- | --- | --- |
| Hành động | Đẩy được thùng hàng | Không đẩy được thùng hàng |
| Lên | ‘U’ | ‘u’ |
| Xuống | ‘D’ | ‘d’ |
| Trái | ‘L’ | ‘l’ |
| Phải | ‘R’ | ‘r’ |

* Hàm tiến triển (successcor): hàm ‘updateState’ nhận tham số là toạ độ hiện tại của player, toạ độ hiện tại của các thùng hàng, hành động hợp lệ sau đó trả về trạng thái tiếp theo.

def updateState(posPlayer, posBox, action) -> newState:

* Hàm kiểm tra mục tiêu (goal test): hàm ‘isEndState’ nhận tham số là toạ độ hiện tại của các thùng hàng và trả về trạng thái hiện tại có phải là trạng thái kết thúc hay không.

def isEndState(posBox) -> bool:

* Trạng thái khởi đầu (initial state): là trạng thái được thiết lập sẵn bởi người làm game, ví dụ ở Hình 1 có trạng thái khởi đầu là ( (5, 5), ( (1, 3), (4, 4) ) ).
* Trạng thái kết thúc (end state): là trạng thái mà khi đó tất cả các thùng hàng được chuyển đến tất cả kho hàng, Ví dụ trong Hình 1, trạng thái kết thúc có thể là

( (xp, yp), ( (1, 3), (1, 6) ) ) hoặc ( (xp, yp), ( (1, 6), (1, 3) ) ), trong đó (xp, yp) là toạ độ của các ô trống bất kì mà player có thể đi tới được.

# 3. Các Thuật Toán

## 3.1 Depth First Search (DFS)

define depthFirstSearch(*gameState*):

moves 🡸 empty array

frontier 🡸 empty stack

Push initial game state into frontier

exploredSet 🡸 empty set

actions 🡸 empty stack

Push a random thing into actions queue #For actions length equal to frontier length

while *frontier is not empty* do

Pop a node from frontier

Pop a nodeAction from actions

if *Is end state* then

moves 🡸 nodeAction

break

if *Node in exploredSet* then

Skip current node

else

Add current node to exploredSet

Get legal actions of the player for the current state

foreach *legal action* do

Find next state

if *Is deadlock* then

Skip current legal action

else

Push new state into frontier stack

Push new action into actions stack

return moves

## 3.2 Breadth First Search (BFS)

define breadthFirstSearch(*gameState*):

moves 🡸 empty array

frontier 🡸 empty queue

Enqueue initial game state into frontier

exploredSet 🡸 empty set

actions 🡸 empty queue

Enqueue a random thing into actions queue #For actions length equal to frontier length

while *frontier is not empty* do

Pop a node from frontier

Pop a nodeAction from actions

if *Is end state* then

moves 🡸 nodeAction

break

if *Node in exploredSet* then

Skip current node

else

Add current node to exploredSet

Get legal actions of the player for the current state

foreach legal action do

Find next state

if Is deadlock then

Skip current legal action

else

Enqueue new state into frontier queue

Enqueue new action into actions queue

return moves

## 3.3 Uninformed Cost Search (UCS)

define uninformedCostSearch(gameState):

moves 🡸 empty array

frontier 🡸 empty priority queue

Enqueue initial game state into frontier with priority 0

exploredSet 🡸 empty set

actions 🡸 empty priority queue

Enqueue a random thing into actions queue #For actions length equal to frontier length

while *frontier is not empty* do

Pop a node from frontier

Pop a nodeAction from actions

if *Is end state* then

moves 🡸 nodeAction

break

if *Node in exploredSet* then

Skip current node

else

Add current node to exploredSet

Get legal actions of the player for the current state

foreach legal action do

Find next state

if *Is deadlock* then

Skip current legal action

else

Get the cost value of new action

Enqueue new state into frontier priority queue with the calculated cost

Enqueue new action into actions priority queue with the calculated cost

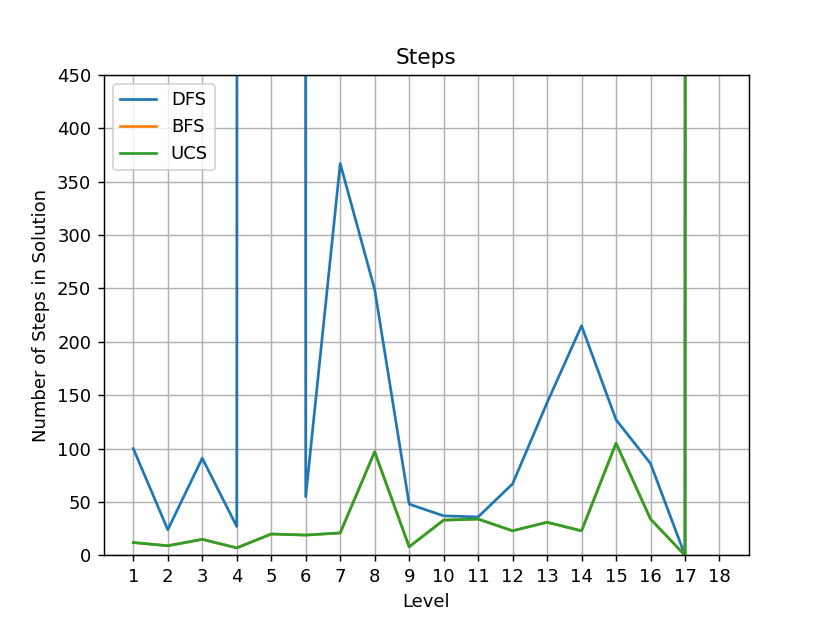
return moves

# 4. Kết Quả

Độ hiệu quả của các thuật toán được đánh giá dựa trên thời gian tìm kiếm, tổng số trạng thái đã duyệt và số lượng bước đi trong kết quả của thuật toán.

Độ hiệu quả của các thuật toán được biểu thị qua các **Hình 2**, **Hình 3, Hình 4, Hình 5**:

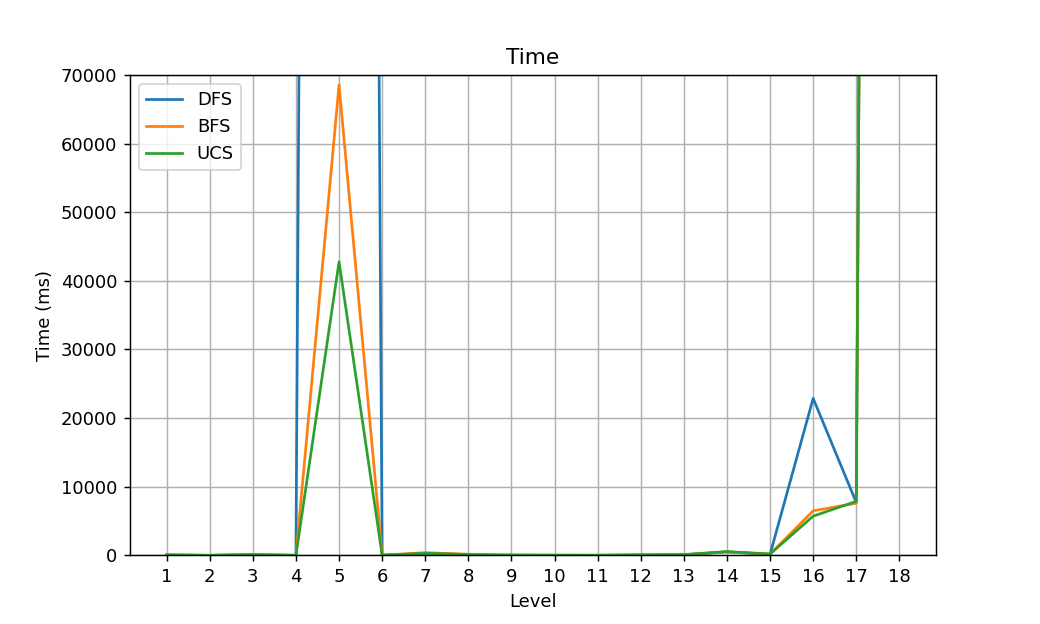
Hình 2: Số bước trong lời giải.



Từ biểu đồ ta có thể thấy DFS không thể giải level 5 và 18, trong khi đó UCS và BFS chỉ không thể tìm ra lời giải cho level 18, DFS cũng cho ra số bước trong lời giải lớn hơn nhiều so với BFS và UCS, do chi phí để chuyển trạng thái là bằng nhau nên BFS và UCS cho ra cùng số bước trong lời giải.

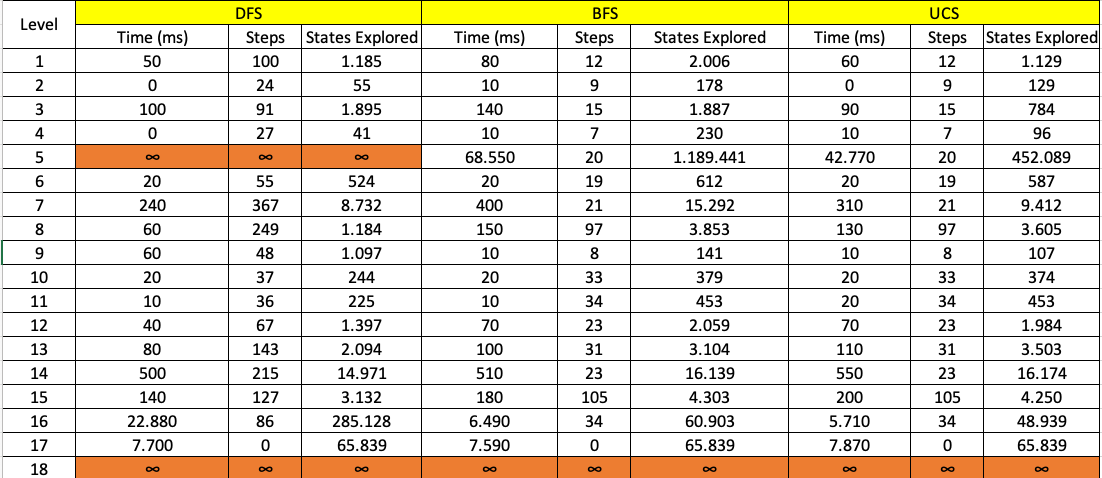
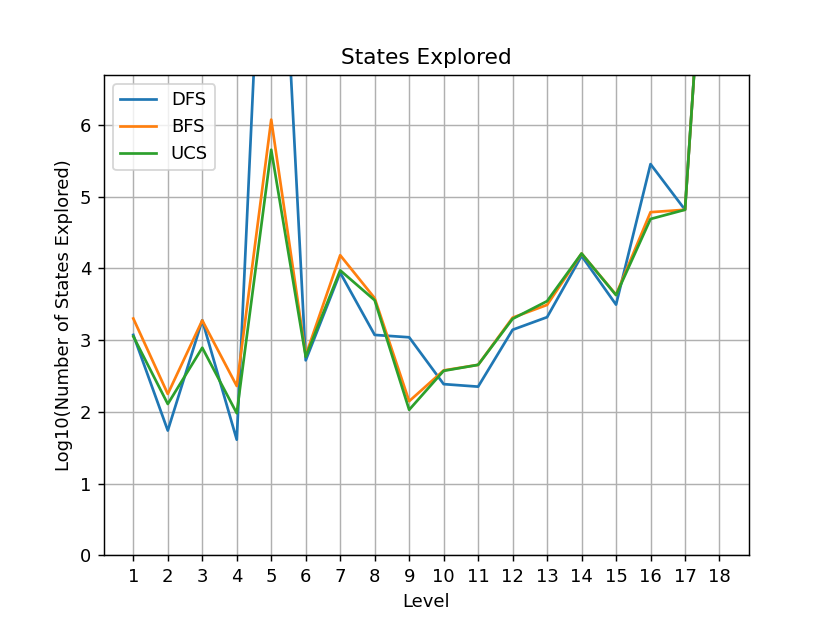
Về thời gian, cả BFS DFS và UCS có thời gian tìm kiếm khá tương đồng ở các level đơn giản, sự khác biệt về thời gian được thể hiện rõ ở các level khó như level 5 và 16, 17. Ở level 5, ta có thể thấy UCS nhanh hơn khoảng 37% so với BFS (42770 ms so với 68550 ms). ở level 16, DFS chậm hơn khoảng 65% so với UCS và BFS.

Hình 3: Thời gian tìm kiếm.



Về số lượng trạng thái đã duyệt, sự cách biệt giữa BFS và UCS là không nhiều ở các level đơn giản, nhưng ở level có độ phức tạp cao như level 5, BFS có số lượng trạng thái đã duyệt cao hơn gần 3 lần so với UCS (1189441 so với 452089), trong khi đó DFS có số trạng thái đã duyệt có thể cao hơn hoặc thấp hơn BFS và UCS trong một vài trường hợp như level 2, 4 hoặc 16.

Hình 4: Log10 của số trạng thái đã duyệt.



Hình 5: Thống kê chi tiết.

# 5. Kết Luận

Qua quá trình cài đặt và chạy các thuật toán DFS, BFS, UCS, ta có thể rút ra các kết luận:

* DFS có số bước trong lời giải lớn hơn tương đối so với BFS và UCS, trong khi đó BFS và UCS có số bước bằng nhau.
* UCS và BFS có thời gian tìm kiếm thấp hơn DFS, DFS có số trạng thái đã duyệt thấp hơn UCS và BFS trong đa số các level 🡺 Không có thuật toán nào là tốt hơn cả, mỗi thuật toán đều có thế mạnh riêng và được sử dụng tuỳ vào bài toán.
* Trong các bản đồ thì bản đồ 18 khó giải nhất vì có số lượng kho hàng lớn + bản đồ nhiều đường gấp khúc nên có độ phức tạp cao.