***Tên: Lương Phạm Bảo***

***MSSV: 19521242***

**BÁO CÁO ASSIGNMENT 1**

1. **Depth Fisrt Search Method (DFS):**

* Mô hình hóa Sokoban:
  + Trong DFS, Sokoban được mô hình hóa thành một cây gồm các node.
  + Mỗi node bao gồm tập các vị trí player và vị trí box từ node parent đến node hiện tại.(là hai yếu tố có vị trí thay đổi)
  + Mỗi node sẽ mở ra các node con của nó là các bước đi hợp lệ mà node hiện tại có thể sinh ra
  + Một stack được tạo ra để lưu trữ các node đang nằm trong frontier và một stack khác để lưu trữ các hành động (actions) của các node nằm trong frontier. Lần lượt lấy phần tử đầu stack ra để tìm các node con hợp lệ của nó và các hành động để đến các node con sau đó push vào frontier và actions.(action là trace-back để khi ta tìm được đường đi ta sẽ dựa vào lộ trình đó để đi)
  + Thuật toán sẽ dừng lại khi một node bị lấy ra khỏi frontier và tìm được một node con là trạng trái kết thúc hoặc frontier rỗng (đã duyệt hết tất cả đường đi có thể nhưng không thể tìm được đường đi).
* Trạng thái khởi đầu: vị trí player, vị trí các box lúc bắt đầu game(beginState).
* Trạng thái kết thúc: các box đã ở đúng vị trí target(goalState).
* Không gian trạng thái: được mở rộng theo chiều dọc, liên tục lấy các node cuối cùng trong frontier (node có độ cao lớn nhất) để mở rộng.
* Các hành động hợp lệ: là hành động không lặp lại một trạng thái đã được ghi nhận và vị trí player không trùng vị trí box, vị trí wall hoặc vị trí box không được trùng vị trí box khác, vị trí wall.
* Hàm tiến triển: node có độ sâu lớn nhất (node ở đầu stack) được ưu tiên mở trước.

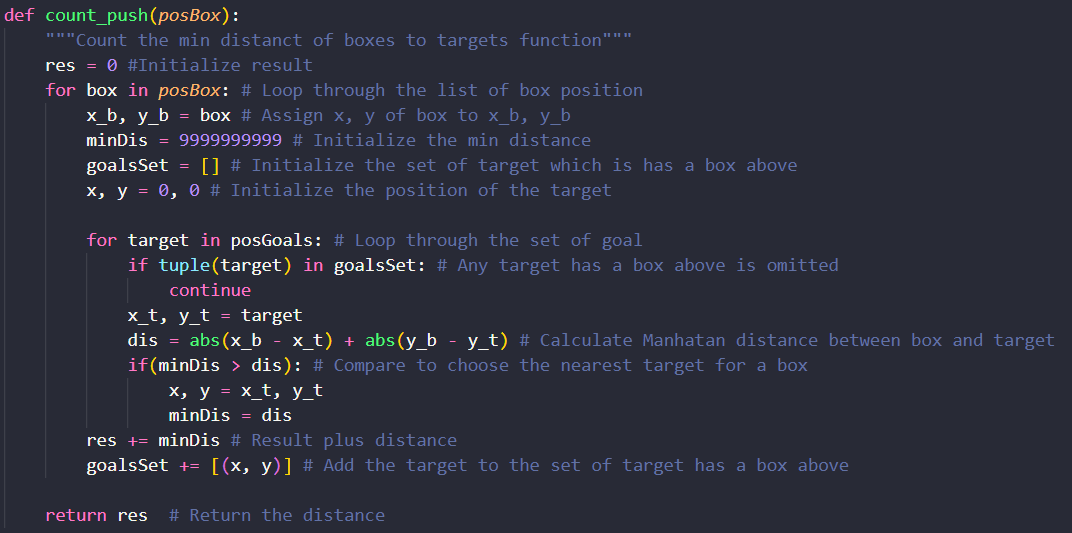
1. **Breadth First Search Method (BFS):**

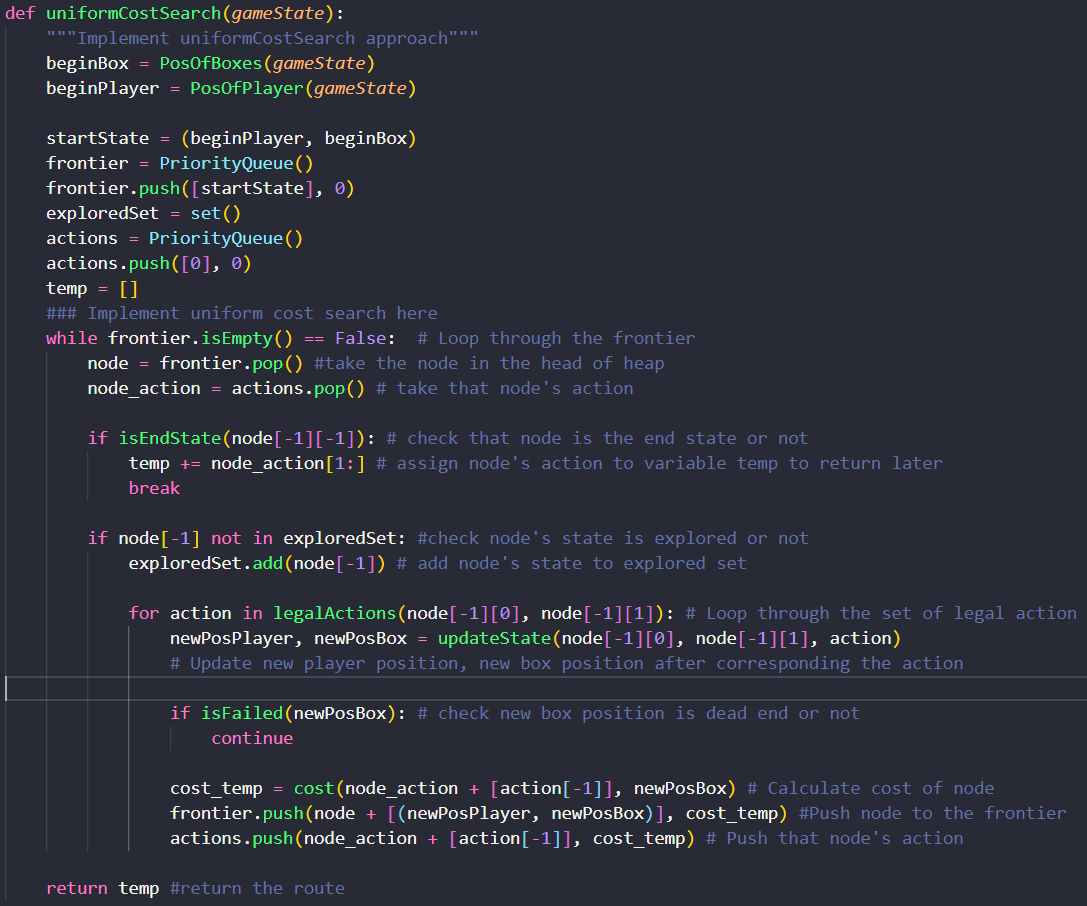
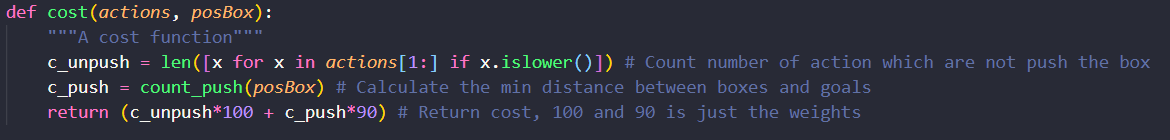
* Mô hình hóa Sokoban:
  + Trong BFS, Sokoban được mô hình hóa thành một cây gồm các node.
  + Mỗi node bao gồm tập các vị trí player và vị trí box từ node parent đến node hiện tại.
  + Mỗi node sẽ có các node con của nó là các bước đi hợp lệ mà node hiện tại có thể sinh ra.
  + Một queue được tạo ra để lưu trữ(vì queue có chi phí để push và pop là O(1)) các node đang nằm trong frontier và một queue khác để lưu trữ các hành động (actions) của các node nằm trong frontier. Lần lượt lấy phần tử đầu queue ra để tìm các node con hợp lệ của nó và các hành động để đến các node con sau đó push vào đầu frontier và actions.
  + Thuật toán sẽ dừng lại khi một node bị lấy ra khỏi frontier và tìm được một node con là trạng trái kết thúc hoặc frontier rỗng (đã duyệt hết các đường đi có thể nhưng không thể tìm được đường đi).
* Trạng thái khởi đầu: vị trí player, vị trí các box lúc bắt đầu game.
* Trạng thái kết thúc: các box đã ở đúng vị trí target.
* Không gian trạng thái: được mở rộng theo chiều ngang, liên tục lấy các node cuối đầu tiên trong frontier (node có độ cao thấp nhất) để mở rộng.
* Các hành động hợp lệ: là hành động không lặp lại một trạng thái đã được ghi nhận và vị trí player không trùng vị trí box, vị trí wall hoặc vị trí box không được trùng vị trí box khác, vị trí wall.
* Hàm tiến triển: node có độ sâu thấp nhất (node ở đầu queue) được ưu tiên mở trước.



1. **Uniform Cost Search Method (UCS):**

* Mô hình hóa Sokoban:
  + Trong UCS, Sokoban được mô hình hóa thành một cây gồm các node.
  + Mỗi node bao gồm tập các vị trí player và vị trí box từ node parent đến node hiện tại.
  + Mỗi node sẽ có các node con của nó là các bước đi hợp lệ mà node hiện tại có thể sinh ra.
  + Một heap được tạo ra để lưu trữ các node đang nằm trong frontier và một heap khác để lưu trữ các hành động (actions) của các node nằm trong frontier. Các node được đưa vào heap dựa trên giá trị cost của chúng (đánh giá dựa trên khoảng cách di chuyển của player và khoảng cách của box so với target), một node có cost càng thấp thì càng nằm gần đầu heap. Lần lượt lấy phần tử đầu heap ra để tìm các node con hợp lệ của nó và các hành động để đến các node con sau đó push vào frontier và actions.
  + Thuật toán sẽ dừng lại khi một node bị lấy ra khỏi frontier và tìm được một node con là trạng trái kết thúc hoặc frontier rỗng (đã duyệt hết các đường đi có thể nhưng không thể tìm được đường đi).
* Trạng thái khởi đầu: vị trí player, vị trí các box lúc bắt đầu game.
* Trạng thái kết thúc: các box đã ở đúng vị trí target.
* Không gian trạng thái: Được mở rộng dần dựa theo node có cost thấp nhất (node nằm ở đầu frontier).
* Các hành động hợp lệ: là hành động không lặp lại một trạng thái đã được ghi nhận và vị trí player không trùng vị trí box, vị trí wall hoặc vị trí box không được trùng vị trí box khác, vị trí wall.
* Hàm tiến triển: node có cost thấp hơn (node ở đầu heap) sẽ được ưu tiên mở rộng trước.





* Nhận xét:
  + Thông thường, DFS sẽ cho kết quả nhanh hơn BFS, tuy nhiên kết quả thường không tối ưu, thậm chí là rất tệ. Mặc dù chỉ mở rộng theo chiều dọc, tuy nhiên DFS ngốn Ram khá nhanh do đường đi của mỗi node dài ra rất nhanh khi không gian bản đồ để thuật toán mở rộng là rất lớn. Map càng lớn thì khả năng giải được của DFS lại càng giảm. Trong trường hợp bộ nhớ có khả năng đáp ứng được lượng node lưu trữ trong 1 nhánh lớn của cây thì DFS thường sẽ nhanh hơn BFS, nếu không thì DFS sẽ dừng trước khi tìm được kết quả do cạn kiệt bộ nhớ.
  + BFS luôn cho kết quả tối ưu cho bài toán này (chi phí khoảng cách là 1) nhưng thường chậm hơn DFS với các bản đồ nhỏ và vừa. Mặc dù BFS ngốn bộ nhớ chậm hơn DFS do mỗi node được thêm vào frontier chỉ tăng 1 lượng nhỏ về độ dài đường đi nhưng xét theo thời gian dài thì DFS sẽ tiết kiệm bộ nhớ hơn do không gian trạng thái cần để mở rộng BFS là rất lớn. Trường hợp tệ nhất của BFS là khi lời giải nằm ngay node lá cuối cùng của cây. Khi đó, BFS sẽ ngốn bộ nhớ hơn DFS rất nhiều do phải lưu trữ một lượng lớn các node.Vì vậy với level 5 mặc dù con người có thể tìm ra đường đi rất nhanh nhưng BFS lại giải quyết rất chậm( do không gian rộng và trống trãi dẫn đến không gian trạng thái mở rộng là rất lớn)
  + Độ hiệu quả của UCS phụ thuộc vào hàm cost. Khi hàm cost càng chính xác thì UCS sẽ giải rất nhanh và tiết kiệm bộ nhớ do chỉ đi vào những hướng có khả năng đem lại kết quả tốt nhất. Tốc độ của UCS rất đáng kinh ngạc khi giải một map lớn như map 5 thì UCS chỉ mất 1.06s để có được lời giải tối ưu. Trong khi đó DFS “tạch” do giới hạn về bộ nhớ, còn BFS thì đưa ra lời giải tối ưu sau tận 327s.
* Trong bài toán này thì UCS sẽ tốt hơn hết nếu ta tìm được một hàm cost đủ tốt.
* So với 2 thuật toán thì đường đi tìm được bởi UCS(nến tồn tại đường đi) sẽ là đường đi tốt nhất về chi phí ,tuy nhiên thuật toán trên sẽ không quan tâm về số độ sâu của cây tìm kiếm mà chỉ quan tâm chi phí đi qua điểm vì vậy nó thật sự không quá hiệu quả với bài toán trên (việc tìm đường đi tối ưu bài toán trên là tương đối khó(vì ta không biết chi phí lẫn độ sâu tối thiểu cần đi để tìm ra được đường đi vì không gian để tìm kiếm quá rộng cũng như bài toán sẽ duyệt hết các đường đi(trong khi BFS có thể dừng lại khi tìm được đường đi
* Với các thuật toán Uniform Search ở trên thì đường đều có ưu và khuyết điểm riêng để giải quyết hiệu quả bài toán trên rằng việc tìm dường đi ở các bản đồ phức tạp hay rộng lớn đều không hiệu quả( tốn nhìu thời gian và không gian tính toán vì đa phần đường đi đều không dẫn tới đích cũng như có nhìu hộp( việc chọn ưu tiên hộp để di chuyển là rất khó khăn và nếu các điểm đặt hộp quá gần thì sẽ dẫn đến việc tìm ra đường đi dễ bị lâm vào bế tắc.