BÁO CÁO BÀI TẬP 1

Huỳnh Thiện Tùng – 19522492

-o-o-o-o-o-o-o-o-o-o-

Depth First Search (DFS):

1. **Mô hình hóa**
   - Xây dựng dựa trên mô hình cây nhị phân gồm các node đại diện tương ứng
   - Mỗi node có thể mở ra để lấy thông tin (vị trí player, vị trí box) và có thể thay đổi
   - Mỗi node có thể mở ra node con, đây là bước đi hợp lệ mà từ node ban đầu có thể đi đến node hiện tại
   - Sử dụng hai stack chứa các node trong frontier và chứa các hành động (actions) của các node trong stack frontier.
   - Duyệt cây bằng cách lấy lần lượt từng node **đầu** trong stack frontier và tìm các node kế tiếp và hành động hợp lệ. Nếu đường đi đó hợp lệ thì push hành động đó vào actions (Khi tìm được lời giải cho bài toán thì actions chính là đường đi để giải)
   - Điều kiện dừng: Khi hang đợi frontier rỗng (không còn node nào để đi qua) hoặc node vừa lấy ra bằng chính trạng thái kết thúc (goal state)
2. **Trạng thái khởi đầu (beginState):** vị trí của các box, ví trí của player
3. **Trạng thái kết thúc (goalState):** vị trí các box đặt đúng
4. **Không gian trạng thái:** duyệt cây theo chiều sâu, ưu tiên mở node nằm ở tầng sâu nhất, mở theo chiều từ trên xuống
5. **Các hành động hợp lệ:** vị trí của player và box không được trùng nhau, không được trùng với wall. Các hành động đi đã được thực hiện để đi đến một trạng thái nhất định thì không được lặp lại
6. **Càm tiến triển (successor function):** ưu tiên mở node có độ sâu lớn nhất (mở node đầu stack)

Breadth First Search (BFS):

1. **Mô hình hóa**
   - Xây dựng dựa trên mô hình cây nhị phân gồm các node đại diện tương ứng
   - Mỗi node có thể mở ra để lấy thông tin (vị trí player, vị trí box) và có thể thay đổi
   - Mỗi node có thể mở ra node con, đây là bước đi hợp lệ mà từ node ban đầu có thể đi đến node hiện tại

- Sử dụng hai hàng đợi queue chứa các node trong frontier và chứa các hành dộng (actions) của các node trong stack frontier. Vì không gian lưu trữ của BFS rất lớn nên việc sử dụng hàng đợi queue để tối ưu chi phí pop và push.
- Duyệt cây bằng cách lấy lần lượt từng node **đầu** trong queue frontier và tìm các node kế tiếp và hành động hợp lệ. Nếu đường đi đó hợp lệ thì push hành động đó vào actions (Khi tìm được lời giải cho bài toán thì actions chính là đường đi để giải)
- Điều kiện dừng: Khi hang đợi frontier rỗng (không còn node nào để đi qua) hoặc node vừa lấy ra bằng chính trạng thái kết thúc (goal state)

2. **Trạng thái khởi đầu (beginState):** vị trí của các box, ví trí của player
3. **Trạng thái kết thúc (goalState):** vị trí các box đặt đúng
4. **Không gian trạng thái:** duyệt cây theo chiều rộng, ưu tiên mở node nằm ở nông nhất, mở theo chiều ngang từ trái qua phải
5. **Các hành động hợp lệ:** vị trí của player và box không được trùng nhau, không được trùng với wall. Các hành động đi đã được thực hiện để đi đến một trạng thái nhất định thì không được lặp lại
6. **Càm tiến triển (successor function):** ưu tiên mở node có độ sâu nhỏ (tầng nông) nhất, chính là lấy phần tử đầu của queue

Uniform Cost Search (UCS):

1. **Mô hình hóa**
   - Xây dựng dựa trên mô hình cây nhị phân gồm các node đại diện tương ứng
   - Mỗi node có thể mở ra để lấy thông tin (vị trí player, vị trí box) và có thể thay đổi
   - Mỗi node có thể mở ra node con, đây là bước đi hợp lệ mà từ node ban đầu có thể đi đến node hiện tại
   - Sử dụng hai cấu trúc dữ liệu **Heap** chứa các node trong frontier và chứa các hành dộng (actions) của các node trong stack frontier. Các node trong Heap được bố trí theo thứ tự cost (chi phí mà player di chuyển và khoảng cách từ box đến dích). Cấu trúc heap cho ta biết rằng, node có chi phí thấp sẽ nằm gần ở đầu heap hơn,
   - Duyệt cây bằng cách lấy lần lượt từng node đầu trong heap frontier và tìm các node kế tiếp và hành động hợp lệ. Nếu đường đi đó hợp lệ thì push hành động đó vào actions (Khi tìm được lời giải cho bài toán thì actions chính là đường đi để giải)
   - Điều kiện dừng: Khi heap frontier rỗng (không còn node nào để đi qua) hoặc node vừa lấy ra bằng chính trạng thái kết thúc (goal state)
2. **Trạng thái khởi đầu (beginState):** vị trí của các box, ví trí của player
3. **Trạng thái kết thúc (goalState):** vị trí các box đặt đúng

4. **Không gian trạng thái:** ưu tiên mở node có chi phí thấp nhất
5. **Các hành động hợp lệ:** vị trí của player và box không được trùng nhau, không được trùng với wall. Các hành động đi đã được thực hiện để đi đến một trạng thái nhất định thì không được lặp lại

6. Càm tiến triển (successor function): ưu tiên mở node có cost thấp, nằm ở đầu heap

Nhận xét:

	DFS	BFS	UCS
Time (s)	Memory limited	179	132s

(Thời gian thực thi của các thuật toán xét trên map phức tạp nhất (map 5) trong sokuban)

- Thông thường, ở các map đơn giản thì DFS chạy nhanh hơn BFS nhưng lời giải của DFS chưa phải là tối ưu. Tuy mở node theo chiều sâu của cây nhưng việc tràn bộ nhớ đôi khi xảy ra nhanh hơn so với BFS. Đối với các map có kích thước lớn và phức tạp thì DFS mặc dù có thể tìm ra lời giải nhưng có thể sẽ đối mặt với việc bị giới hạn bộ nhớ nhanh hơn là BFS.
- Đối với BFS, luôn cho lời giải tối ưu hơn DFS trong các map không quá lớn và không quá phức tạp. Thời gian thực thi của BFS chậm hơn DFS. Việc mở các node theo độ sâu thấp nhất có thể giảm thiểu việc tràn bộ nhớ nhanh so với DFS nhưng nếu xét về tính tổng quát là chạy lâu dài thì BFS thật sự tốn nhiều không gian lưu trữ hơn là DFS. Trường hợp xấu nhất của BFS là khi nằm ở tầng cuối cùng của cây thì thật sự đây là sự thảm hại của thuật toán, thậm chí là rất tệ so với DFS.
- Qua thực nghiệm cho thấy UCS cho kết quả tốt hơn DFS và BFS. Lời giải của UCS luôn tối ưu và thời gian thực thi nhanh hơn so với 2 thuật toán kia. Nhưng nhược điểm lớn nhất của thuật toán này là không quan tâm đến độ sâu của node. Nó chỉ quan tâm đến chi ví tính toán từng hành động. Điều này thật sự rất khó tối ưu cho thuật toán. Ví dụ khi thuật toán đã tìm được lời giải tối ưu nhưng nó vẫn chạy và tìm tiếp lời giải khác, nó không biết dừng lại đúng lúc nhưng BFS thì lại có ưu điểm này. Ngoài ra, để tối ưu thời gian chạy, có thể cải tiến hàm cost, vì trong thuật toán mỗi hành động đều tính chi phí, nếu hàm tính chi phí không tối ưu thì cũng sẽ ảnh hưởng đến thời gian thực thi của thuật toán.

⇨ **Tóm lại:** Cả 3 thuật toán trên đều có ưu nhược điểm khác nhau, mỗi bài toán phụ thuộc vào từng trường hợp cụ thể. Tuy nhiên, xét công bằng trong bài toán Sokuban thì thuật toán UCS có vẻ tốt hơn so với các thuật toán trên. Ngoài ra trong thực tế, chúng ta cần cân nhắc lựa chọn thuật toán phù hợp với ngữ cảnh đang giải quyết vấn đề cho phù hợp.