BÁO CÁO BÀI TẬP LỚN

TRÍ TUỆ NHÂN TẠO

Giải bài toán N-Puzzle sử dụng thuật toán BFS và A\*

Các thành viên:

1. Trần Văn Thắng – 621079
2. Lưu Văn Hưng – 621093
3. Trần Thanh Bình – 621057

Lời nói đầu

Chọn đề tài này với mong muốn có thể áp dụng những kiến thức đã được học để có thể xây dựng 1 ứng dụng cụ thế. Việc này giúp chúng em hiểu sâu hơn về thuật toán, để xử lý những bài toán tìm kiếm trong học tập, nghiên cứu và các công việc sau này.

A. Thuật toán A\*

I. Tìm hiểu thuật toán

Thuật toán A\* là 1 trường hợp của Best First Search nên ta sẽ đi tìm hiểu trực quan BFS trước khi vào A\* để có thể có 1 cái nhìn chung về những thuật toán tìm kiếm tối ưu này.

1. **Khái niệm**

1.1 **Thuật toán tìm kiếm tối ưu – Best First Search**

Thuật toán tìm kiếm ưu tiên tối ưu là sự kết hợp cả 2 thuật toán tìm kiếm theo chiều sâu và tìm kiếm theo chiều rộng, cho phép người dùng đi 1 con đường duy nhất tại 1 thời điểm, nhưng đồng thời vẫn xem xét các hướng đi khác. Nếu nhận thấy rằng hướng đang đi không có triển vọng bằng những con đường còn lại thì ta sẽ chuyển theo 1 trong những con đường đó.

Tại mỗi bước cảu thuật toán BFS ta chọn theo trạng thái có khả năng cao nhất trong số các trạng thái đã được xét cho đến thời điểm đó, như vậy với phương pháp này ta ưu tiên đi vào những nhánh tìm kiếm có khả năng cao nhất.

Tư tưởng chủ đạo của Best First Search giúp ta sẽ không bị đi những vòng luẩn quẩn bởi vì nếu đi càng sâu vào một hướng mà ta phát hiện hướng đi càng ngày càng tệ, thậm chí còn tệ hơn cả những hướng mà ta chưa đi thì sẽ không tiếp tục trên hướng đi hiện tại nữa mà sẽ lựa chọn theo một trong những hướng đi tốt nhất trong tập hợp chưa đi.

**a.** Cài đặt giải thuật BFS.

Heap: Tập hợp biến động các đối tượng theo một thứ tự ưu tiên nhất định mỗi đối tượng được đặc trưng bởi 1 khóa thể hiện mức độ ưu tiên của nó, thông thường các đối tượng được thao tác để chọn ra đối tượng có độ ưu tiên cao nhất đê xử lý tiếp tục.

Heap là cấu trúc dữ liệu trừu tượng, với 4 thao tác cơ bản là đưa một phần tử mới vào trong Heap (Insert), lấy phần tử có độ ưu tiên lớn nhất (Remove), đưa nút có khóa K về đúng vị trí theo chiều hướng đến gốc cây (UnHeap), và đưa nút có khóa K về đúng vị trí theo chiều hướng đến ngọn (các nút lá) (DownHeap).

* Cài đặt giải thuật BFS.

Để cài đặt giải thuật BFS, ta thường dùng 2 tập hợp:

* **OPEN:** Tập hợp những trạng thái đã được sinh ra nhưng chưa được xét đến, OPEN là một hàng đợi ưu tiên mà trong đó phần tử có độ ưu tiên cao nhất là phần tử tốt nhất. Hàng đợi ưu tiên này thường được cài đặt bằng Heap.
* **CLOSE:** Tập chứa các trạng thái đã được xét đến, ta cần lưu trữ những trạng thái này trong bộ nhớ để phòng trường hợp khi một trạng thái mới sinh ra trong quá trình duyệt lại trùng hợp với trạng thái đã xét trước đó. Khi không gian tìm kiếm có dạng cây thì ta không cần sử dụng với tập này
* **Cài đặt cách khác.**

+ Đặt OPEN chứa trạng thái khởi đầu.

Cho đến khi tìm được trạng thái đích hoặc không còn nút nào trong tập OPEN vòng lặp thực hiện duyệt các trạng thái tiếp theo, mỗi bước:

+ Chọn trạng thái tốt nhất Tbest trong OPEN ( và xóa trạng thái Tbest khỏi OPEN)

+ Nếu Tbest là trạng thái kết thúc thì thoát.

+ Nếu Tbest không phải là trạng thái tốt nhất, tạo các nút trạng thái Tn con của Tbest. Đối với mỗi trạng thái kế tiếp, thực hiện tính hàm chi phí f(Tn) và thêm Tn vào trong OPEN

1.2. **Thuật toán tìm kiếm A\***

a. **Khái niệm**

Không giống như thuật toán Dijkstra, A\* là một thuật toán tìm đường đi từ điểm tới điểm, từ một nút khởi đầu đến một nút đích cho trước hoặc tới một thỏa mãn điều kiện và không sử dụng để tìm giải pháp cho vấn đề tìm đường đi ngắn nhất trong học thuyết đồ thị.

Về bản chất, thuật toán này làm việc rất gióng như Dijkstra. Nhưng hiệu quả hơn khi A\* sử dụng một hàm đánh giá bổ sung ( đánh giá heuristic ) để ước lượng vè tuyến đường đi tốt nhất qua nút đó ( Việc lựa chọn trạng thái tiếp theo sẽ phụ thuộc vào hàm lượng giá này ).

A\* Là một ví dụ cho thuật toán **Best first search**, là một thuật toán đầy đủ, luôn tìm thấy một lời giải nêú bài toán có lời giải.

b. **Đặt ván đề**:

Vấn đề đưa ra giống với vấn đề được giải quyết bởi thuật toán tìm đường đi ngắn nhất Dijkstra. Cho một đồ thị ( đồ thị có hướng và trọng số âm ) và hai note trên đồ thị ( gọi là điểm bắt đầu và điểm cuối, cần tìm một con đường có chi phí đi từ đầu tới cuối là nhỏ nhất trong các đường đi có thể, kết quả là danh sách các node có cần đi qua để tới đích với chi phí nhỏ nhất.

c. **Thuật toán**

Hai tập OPEN và CLOSE cũng giống như Best First Search

Có thể nói rằng, thuật toán A\* hoạt động tương tự như Dijkstra đã thực hiện. Hơn thế tại mỗi bước luôn cân nhắc chọn các nút có chi phí thấp nhất trong tập OPEN để định hướng cho bước tiếp theo, chọn nút có nhiều khả năng sẽ dẫn đến đường đi ngắn nhât cần tìm. Khai niệm “ nhiều khả năng ” này được đánh giá bằng một hàm heuristic. Nếu hàm đánh giá càng chính xác thì thuật toán càng tối ưu. Nếu hàm đánh giá tồi thì sẽ dẫn tới thuật toán thực hiện thậm chí tồi tệ hơn cả Dijkstra. Cụ thể hơn, A\* có thể hoạt động trong một vòng lặp ( xét lại các trạng thái đã xét trước đó ). Tuy sử dụng thuật toán lựa chọn tương tự của Dijkstra nhưng với sự khác biệt vè hàm heuristic A\* sẽ hoạt động hiệu quả hơn.

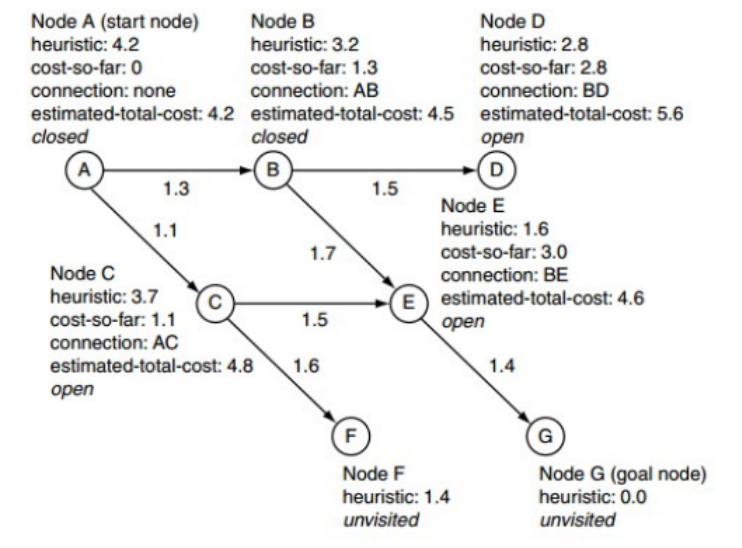
d. **Độ tốt của một lời giải trong thuật toán A\***

Độ tốt của một trạng thái được tính ra dựa trên hai giá trị mà ta gọi là h(n) và g(n).

h(n) là một ước lượng về chi phí từ trạng thái hiện tại cho đến trạng thái đích.

g(n) là chiều dài quãng đường đã đi từ trạng thái ban đầu tới trạng thái đang xét

Một điểm cần lưu ý rằng **g** là chi phí thực sự còn **h**  là chi phí ước lượng. dưới đây là hình ảnh mô tả khái quát về thuật toán, chỉ ra cách tính giá trị cho một số node trên đồ thị



Kết hợp g(n) và h(n) ta có hàm tỏng chi phí ước lượng cho quãng đường đi từ nút xuất phát tới nút đích theo đường đi của trạng thái hiện tại.

Công thức lượng:  **f(n)=g(n)+h(n)**

Nếu giá trị **f(n)** càng thấp thì độ ưu tiên của **n** càng cao.

Hai gia trị này đều quy ước là không âm để thuận tiện cho qúa trình tính toán.