Báo cáo đồ án #1

SEARCH IN AI

Cơ sở trí tuệ nhân tạo - Lớp 18_21

Sinh viên: Nguyễn Bảo Long - MSSV: 18120201

TP Hồ Chí Minh, ngày 27/10/2020

Contents

1 Thông tin cá nhân

1. Họ tên: Nguyễn Bảo Long
2. MSSV: 18120201
3. Email: baolongnguyen.mac@gmail.com

2 Tự đánh giá

STT	Tiêu chí	Điểm
1	Cài đặt thành công thuật toán	10
2	Chọn testcase cho thuật toán (bao gồm TH đặc biệt và TH thông thường)	5
3	Phân tích ý tưởng thuật toán	10
4	Đánh giá thuật toán	10
5	Nhận xét thuật toán	8
Trung bình		8.5

Table 1: Bảng tự đánh giá kết quả làm việc

3 Lý thuyết cơ bản về các thuật toán trong báo cáo

3.1 Thuật toán DFS

- Ý tưởng
  - Depth-first Search mở rộng tìm kiếm tại các node ở "sâu nhất" trong tập hợp các node đang "mở". Nghĩa là thuật toán sẽ đi vào 1 nhánh của đồ thị xa nhất có thể trước khi quay lui
  - Depth-first Search sử dụng 1 stack với nguyên lý LIFO trong quá trình mở rộng
- Đánh giá thuật toán
  - Gán
    - d = Độ sâu của cây tìm kiếm
    - n = Số nhánh tối đa của 1 node
      $\Rightarrow n^i$ = Số node tại độ cao i (giả sử cây tìm kiếm là cây đầy đủ)
  - Độ phức tạp về thời gian: $T(n) = 1 + n^2 + n^3 + ... + n^d = O(n^d)$
  - Độ phức tạp về không gian: $S(n) = O(nd)$
  - Tính đầy đủ: Thuật toán được hình thành theo tư duy vét cạn, nghĩa là khám phá mọi trường hợp có thể xảy ra. Do đó, nó có thể trả về lời giải cho bài toán nếu lời giải đó tồn tại
  - Tính tối ưu: Thuật toán không cho lời giải tối ưu vì chưa chắc tìm ra được đường đi ngắn nhất cho bài toán

3.2 Thuật toán BFS

- Ý tưởng
  - Breadth-first Search mở rộng tìm kiếm tại các node ở "độ sâu nhỏ nhất" trong tập hợp các node đang "mở". Nghĩa là thuật toán sẽ đi vào tất cả các node kề với node hiện tại
  - Bài toán sử dụng 1 queue với nguyên lý FIFO trong quá trình mở rộng
- Đánh giá thuật toán
  - Gán
    - d = Độ sâu của cây tìm kiếm
    - s = Độ sâu của trạng thái đích
    - n = Số nhánh tối đa của 1 node
      $\Rightarrow n^i$ = Số node tại độ cao i (giả sử cây tìm kiếm là cây đầy đủ)
  - Độ phức tạp về thời gian: $T(n) = 1 + n^2 + n^3 + ... + n^s = O(n^s)$
  - Độ phức tạp về không gian: $S(n) = O(n^s)$ - kích thước của hàng đợi tại độ sâu s
  - Tính đầy đủ: Thuật toán được hình thành theo tư duy vét cạn, nghĩa là khám phá mọi trường hợp có thể xảy ra. Do đó, nó có thể trả về lời giải cho bài toán nếu lời giải đó tồn tại
  - Tính tối ưu: Thuật toán sẽ đưa ra lời giải với ít trạng thái trung gian nhất

3.3 Thuật toán UCS

- Ý tưởng
  - Khởi tạo mảng d: $d[node_i]$ = chi phí đường đi tính từ trạng thái khởi tại đến trạng thái $node_i$. Ban đầu, $d[node_i]$ được gán bằng giá trị MAX_INT với mọi i
  - Thuật toán chọn mở rộng vào một node nếu node đó thoả mãn $d[node] = min$
  - Cập nhật lại chi phí đường đi tính từ trạng thái khởi tạo sau mỗi lần mở rộng đến các node kề
- Đánh giá thuật toán
  - Gán
    - C^* = Chi phí của lời giải tối ưu
    - ϵ = Chi phí tối thiểu tại mỗi bước
  - Độ phức tạp về thời gian $T(n) = O(n^{C^*/\epsilon})$
  - Độ phức tạp về không gian $S(n) = O(n^{C^*/\epsilon})$
  - Tính đầy đủ: Thuật toán được hình thành theo tư duy vét cạn, nghĩa là khám phá mọi trường hợp có thể xảy ra. Do đó, nó có thể trả về lời giải cho bài toán nếu lời giải đó tồn tại
  - Tính tối ưu: Thuật toán trả về lời giải tối ưu nhất cho bài toán

3.4 Thuật toán AStar

- Ý tưởng
  - Thuật toán mở rộng các node được cho là gần trạng thái đích nhất
  - Thuật toán đánh giá một node là gần hoặc xa so với trạng thái đích bằng hàm **heuristic**
  - Hàm **heuristic** của thuật toán được xây dựng như sau
    - $f(n) = h(n) + g(n)$
    - $g(n)$ = Chi phí để tới được node n
    - $h(n)$ = Chi phí ước tính từ node n tới trạng thái đích
- Đánh giá thuật toán
  - Độ phức tạp về thời gian: Phụ thuộc vào heuristic được chọn. Trong trường hợp tệ nhất (không gian tìm kiếm vô tận), độ phức tạp về thời gian của thuật toán có thể lên tới $O(b^d)$
    - Gọi b_i là số cây con của một node ứng với hàm heuristic h_i, d là độ sâu của trạng thái đích trong cây tìm kiếm
    - Ta có: $T(n) = 1 + b_i^2 + b_i^3 + ... + b_i^d$
    - Một heuristic tốt cho phép giảm giá trị b xuống tối thiểu bằng 1
    - Người ta chứng minh được rằng hàm heuristic chấm nhận được nằm trong khoảng
    $$0 < h(x) \leq h(x)^*$$
    - Trong đó, $h(x)$ là hàm heuristic được chọn, $h(x)^*$ là chi phí thực tế

- Độ phức tạp về không gian: Tương tự thuật toán UCS
- Tính đầy đủ: Trong mọi đồ thị, nếu tồn tại một đường đi từ trạng thái ban đầu đến trạng thái đích thì thuật toán sẽ tìm được đường đi đó
- Tính tối ưu: Nếu heuristic được chọn thoả mãn điều kiện nêu trên thì thuật toán chưa chắc cho ra lời giải tối ưu nhất nhưng đó là một trong số những lời giải tối ưu

3.5 Thuật toán Dijkstra

- Ý tưởng:
  - Thuật toán sử dụng một mảng d với $d(node_i)$ = đường đi ngắn nhất tính từ trạng thái khởi tạo đến trạng thái $node_i$. Do đó, ta có công thức như sau

$$d(node_i) = min\{d(node_j) + w(node_i, node_j), node_j \in Adj(node_i)\}$$

  - Trong đó, $w(node_i, node_j)$ là chi phí đi từ $node_i$ đến $node_j$; $Adj(node_i)$ là tập hợp các dỉnh kề với $node_i$
  - Thông thường khi cài đặt, ta thường gán giá trị khởi tạo cho mảng d là vô cùng. Sau đó dần dần thay thế bằng các giá trị thức tế của đồ thị. Node tiếp theo được chọn để mở rộng là node có chi phí nhỏ nhất trong mảng d
- Đánh giá thuật toán
  - Độ phức tạp về thời gian: $O(n^2)$ với n là số đỉnh của đồ thị (Vì phải duyệt qua 2 lần tất cả các đỉnh của đồ thị trong 2 vòng lặp lồng nhau)
  - Độ phức tạp về không gian: $O(n)$ với n là số đỉnh của đồ thị (Vì chỉ lưu trữ lại chi phí ngắn nhất đi từ trạng thái khởi tạo đến các đỉnh của đồ thị)
  - Tính đầy đủ: Thuật toán được hình thành theo tư duy vét cạn, nghĩa là khám phá mọi trường hợp có thể xảy ra. Do đó, nó có thể trả về lời giải cho bài toán nếu lời giải đó tồn tại
  - Tính tối ưu: Lời giải bài toán đưa ra là tối ưu nhưng chi phí để tìm lời giải trong trường hợp bài toán có không gian trạng thái lớn là rất cao

3.6 Thuật toán Greedy

- Ý tưởng
  - Thuật toán sử dụng hàm heuristic "$h(n)$ = Ước lượng chi phí/khoảng cách từ n đến đích" trong việc mở rộng tìm kiếm của mình
  - Xét tại cục bộ (bao gồm node hiện tại và các node kề nó trong đồ thị), Greedy chọn đường đi có $h(n)$ là nhỏ nhất
- Đánh giá thuật toán
  - Độ phức tạp về thời gian trong trường hợp tệ nhất: $O(b^m)$ với m là độ sâu tối đa của cây tìm kiếm. Có thể cải thiện bằng cách chọn hàm heuristic tốt hơn
  - Độ phức tạp về không gian trong trường hợp tệ nhất: Tương tự độ phức tạp về thời gian

- Tính đầy đủ: Thuật toán có thể không tìm được câu trả lời cho bài toán. Trong trường hợp quản lý không tốt các trạng thái đã đi qua, Greedy có thể rơi vào tình huống vòng lặp vô tận
- Tính tối ưu: Thuật toán không quan tâm đến chi phí tính từ trạng thái khởi tạo đến trạng thái hiện tại mà chỉ quan tâm đến chi phí/khoảng cách từ trạng thái tiếp theo đến trạng thái đích nên chỉ cho lời giải tối ưu trong trường hợp chi phí giữa các trạng thái là như nhau.

4 Điểm khác nhau giữa UCS và AStar

Điểm khác nhau dễ thấy nhất giữa 2 thuật toán UCS và A* là sự có mặt của hàm heuristic $h(node)$ trong thuật toán A*. Hàm này cung cấp cho thuật toán một thông tin quan trọng: Tại $node_i$, $h(node_i)$ ước lượng chi phí/khoảng cách từ $node_i$ đến trạng thái đích. Do đó, ta phân A* vào nhóm các thuật toán tìm kiếm có thông tin còn UCS được phân vào nhóm các thuật toán tìm kiếm mù.

Các điểm khác nhau còn lại có thể kể đến như

- Độ phức tạp về thời gian: Trong trường hợp chọn được hàm heuristic chấp nhận được, độ phức tạp về thời gian của thuật toán A* có thể thấp hơn nhiều lần so với thuật toán UCS
- Ý tưởng tìm đường đi của 2 thuật toán đều dựa vào cùng 1 hàm $f(n)$ nhưng cách triển khai hàm $f(n)$ của 2 thuật toán là khác nhau
  - UCS sử dụng hàm chi phí $g(n)$ làm tiêu chí chọn nước đi tiếp theo. Do đó ta được $f_{UCS}(n) = g(n)$
  - A* sử dụng tổng của hàm chi phí và hàm heuristic làm tiêu chí chọn nước đi tiếp theo. Do đó ta được $f_{A*}(n) = h(n) + g(n)$

5 Chi tiết cài đặt

5.1 Testcase chung

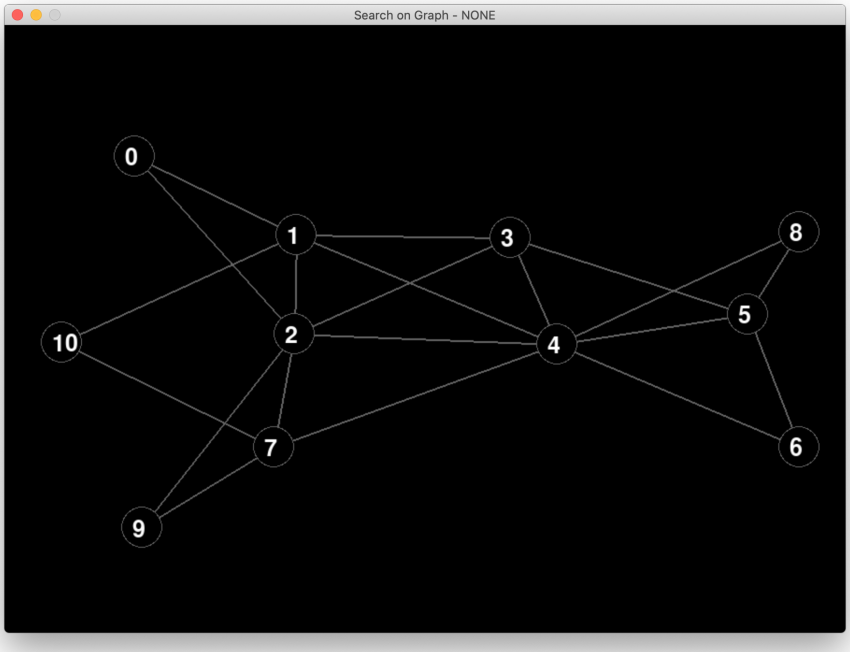

Figure 1: Testcase chung cho các thuật toán (testcase1)

5.2 Thuật toán DFS

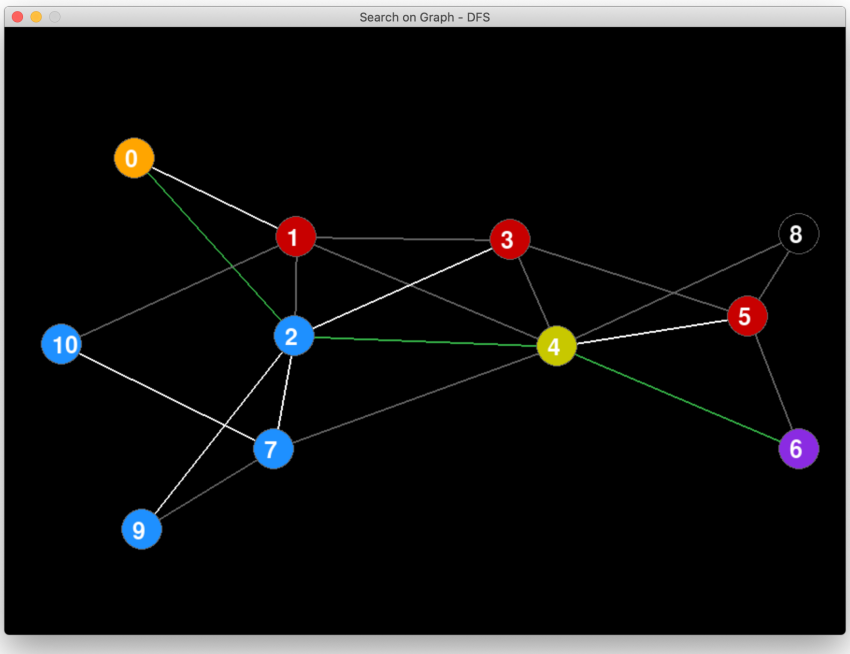

Figure 2: Kết quả tesecase1 của thuật toán DFS

- Thuật toán không quan tâm đến chi phí đường đi
- Không có thông tin về trạng thái đích trong quá trình tìm kiếm
- Có thể khắc phục tính đầy đủ của thuật toán bằng cách đặt giới hạn độ sâu trong giải thuật

5.3 Thuật toán BFS

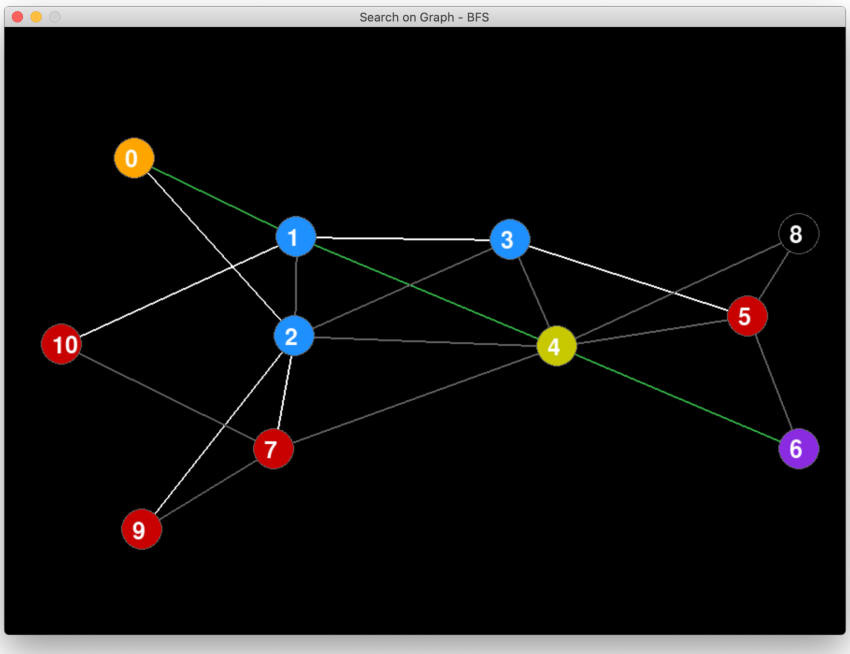

Figure 3: Kết quả tesecase1 của thuật toán BFS

- Thuật toán không quan tâm đến chi phí đường đi
- Không có thông tin về trạng thái đích trong quá trình tìm kiếm
- Cách sử dụng bộ nhớ của thuật toán không hiệu quả bằng DFS

5.4 Thuật toán UCS

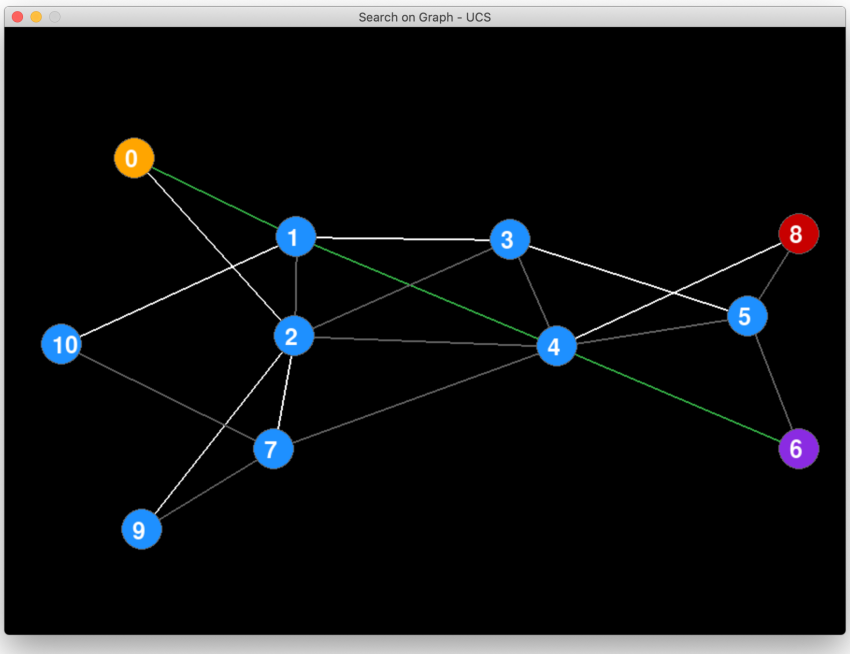

Figure 4: Kết quả tesecase1 của thuật toán UCS

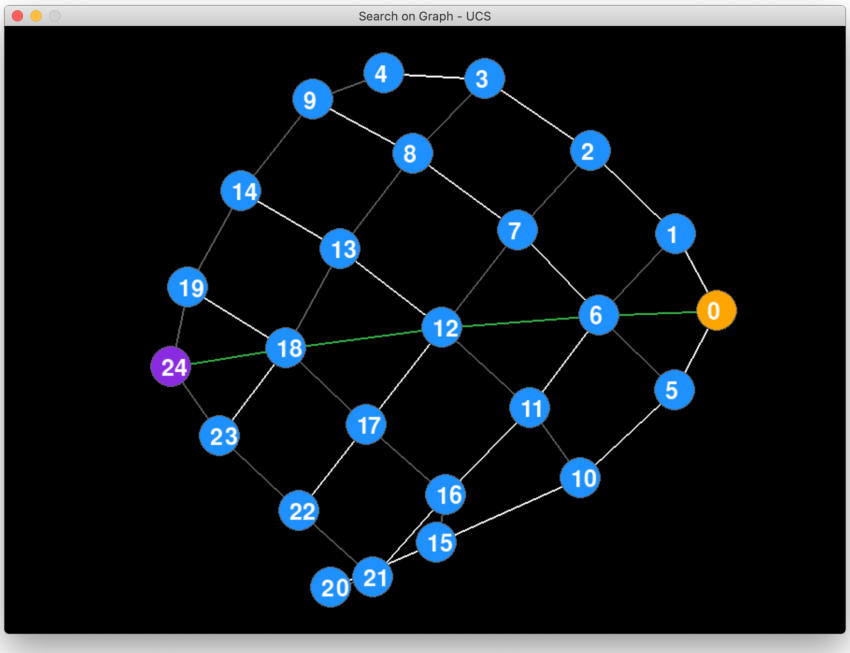

Figure 5: Kết quả tesecase2 của thuật toán UCS

- Không có thông tin về trạng thái đích trong quá trình tìm kiếm
- Trong một vài trường hợp, độ phức tạp của thuật toán có thể lớn hơn độ phức tạp của BFS

5.5 Thuật toán AStar

- Thuật toán chạy có tốt hay không phụ thuộc hoàn toán vào cách chọn hàm heuristic
- Có 3 loại khoảng cách cần cân nhắc khi chọn hàm heuristic cho thuật toán
  - Khoảng cách Manhattan

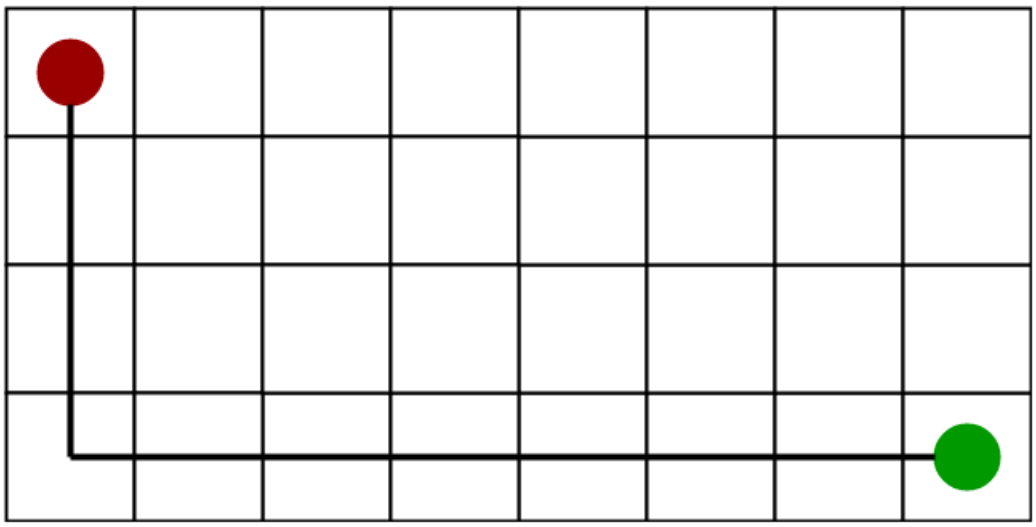

Figure 6: Khoảng cách Manhattan

    * Áp dụng cho trường hợp chỉ có thể di chuyển theo 4 hướng (trái, phải, trên, dưới)
    * $h = |current\_cell.x - goal\_cell.x| + |current\_cell.y - goal\_cell.y|$
  - Khoảng cách Diagonal

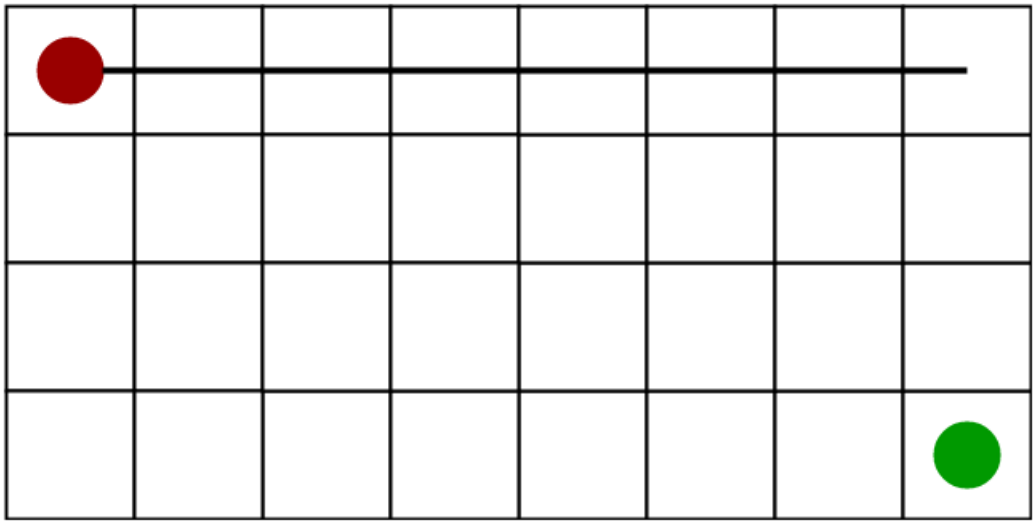

Figure 7: Khoảng cách Diaginal

    * Áp dụng cho trường hợp có thể di chuyển theo 8 hướng (trái, phải, trên, dưới và đường chéo)

    * $h = \{|current\_cell.x - goal\_cell.x|, |current\_cell.y - goal\_cell.y|\}$
  - Khoảng cách Euclidean

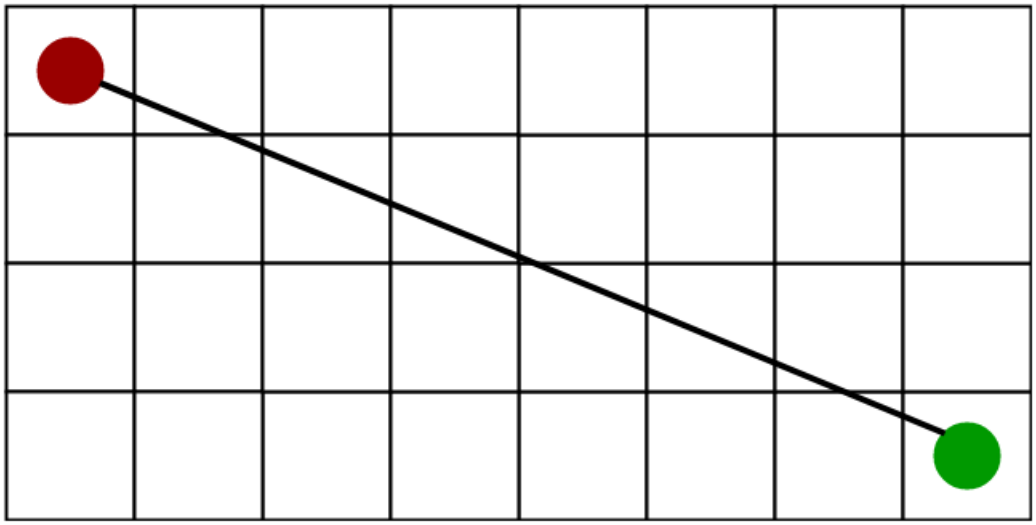

Figure 8: Khoảng cách Euclidean

    * Áp dụng cho trường hợp có thể di chuyển theo hướng bất kỳ
    * $h = \sqrt{(current\_cell.x - goal\_cell.x)^2 + (current\_cell.y - goal\_cell.y)^2}$

- Vì bài toán hiện tại có thể di chuyển theo hướng bất kỳ (miễn là thoả mãn đồ thị) nên hàm heuristic được chọn là khoảng cách Euclidean
- Testcase

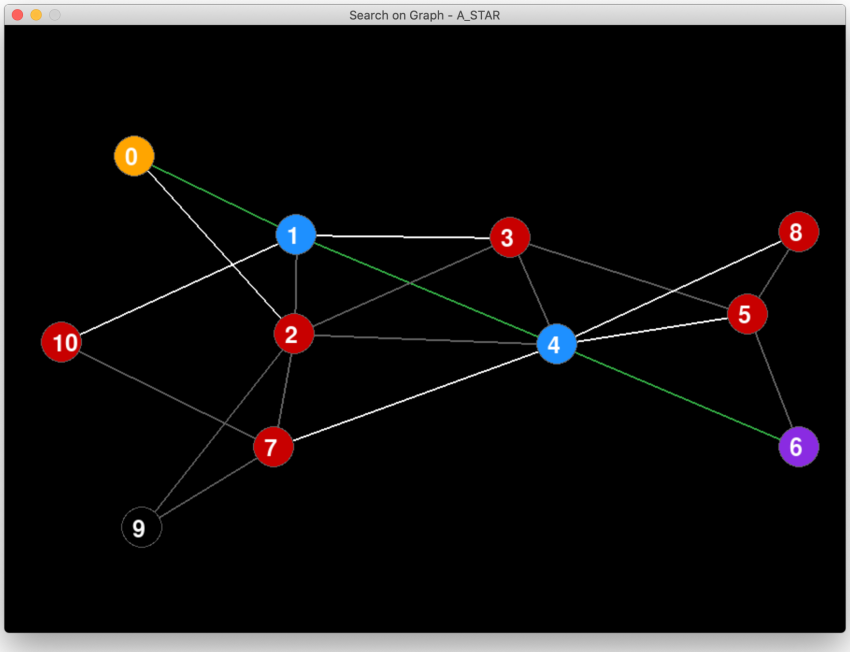

Figure 9: Kết quả tesecase1 của thuật toán A*

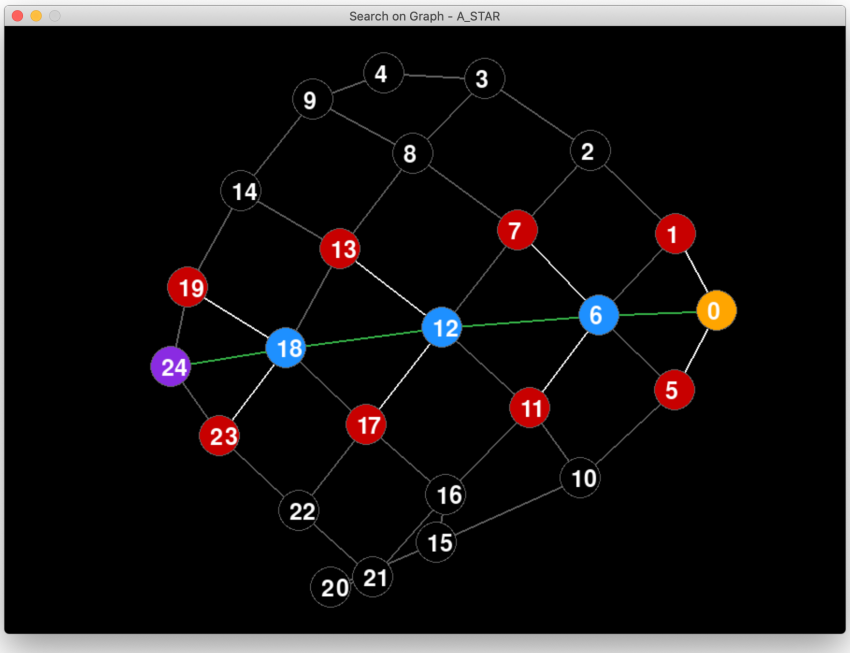

Figure 10: Kết quả tesecase2 của thuật toán A*

5.6 Thuật toán Dijkstra

- Dijkstra tìm đường đi ngắn nhất từ trạng thái đầu đến toán bộ trạng thái có trong không gian trạng thái. Do đó, thuật toán này là không khả thi trong trường hợp không gian trạng thái rất lớn
- Có thể thấy được trong 2 testcase bên dưới, thuật toán mở rộng toàn bộ đồ thị để có thể

tìm ra được đường đi ngắn nhất

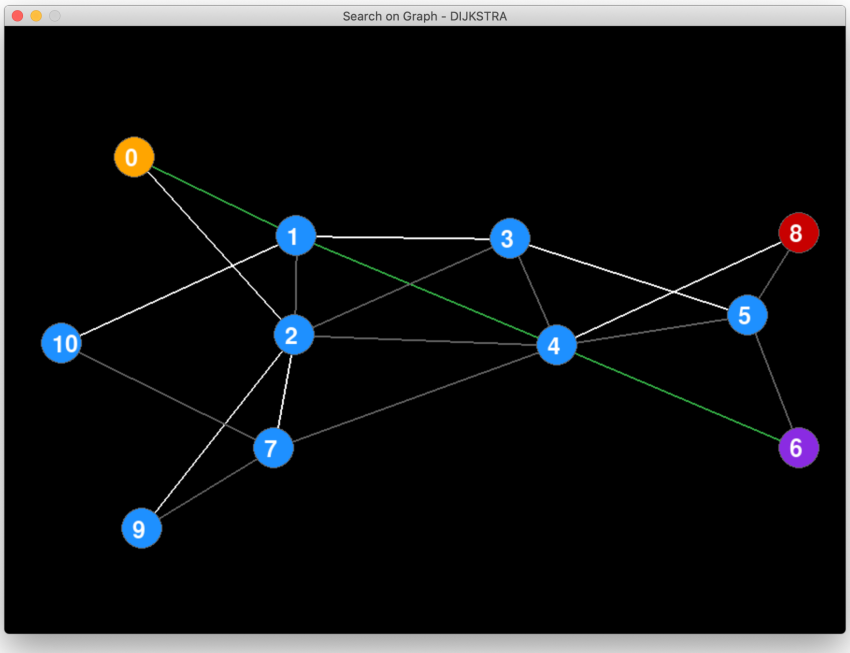

Figure 11: Kết quả tesecase1 của thuật toán Dijkstra

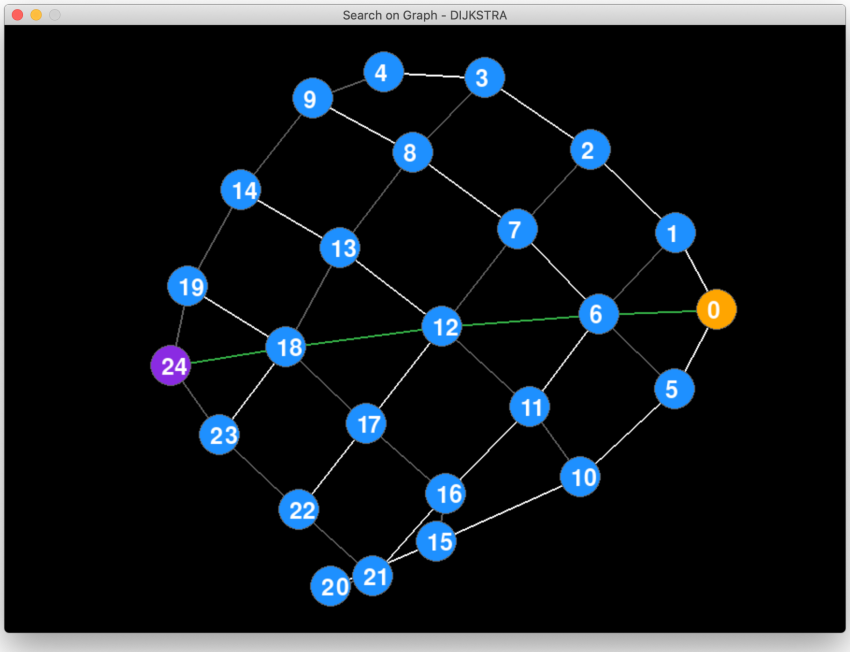

Figure 12: Kết quả tesecase2 của thuật toán Dijkstra

5.7 Thuật toán Greedy

- Do vật, thuật toán có xu hướng cho ra kết quả trong thời gian nhanh nhất nhưng không phải lúc nào cũng là tốt nhất

- Thuật toán không quan tâm đến chi phí tính từ trạng thái khởi tạo đến trạng thái hiện tại mà chỉ quan tâm đến việc node tiếp theo gần đích hay không
- Trong trường hợp xấu, vì tính tham lam của mình, thuật toán có thể đi vào ngõ cụt. Trường hợp này không được minh hoạ trong testcase do giới hạn của người cài đặt và tính ngẫu nhiên trong việc chọn vị trí vẽ node trên đồ thị của source thuật toán
- Sự tham lam này khác với cách chọn node để đi tiếp của thuật toán Hill Clamping (xét trên cùng 1 heuristic)
  - Thuật toán Hill Clamping chọn node tiếp theo trong tập hợp các node kề. Điều kiện chọn là node tiếp theo đó phải tốt nhất trong tất cả các node kề và phải tốt hơn trạng thái hiện tại. Nếu không chọn ra được node thoả tính chất đó, thuật toán sẽ kết luận không tìm được lời giải bất chấp việc bài toán có tồn tại lời giải hay không
  - Thuật toán Greedy cũng chọn vị trí tốt nhất ở cục bộ để đi tiếp nhưng dù cho vị trí đi tiếp đó có tệ hơn vị trí hiện tại thì thuật toán vẫn cứ đi vào
- Testcase

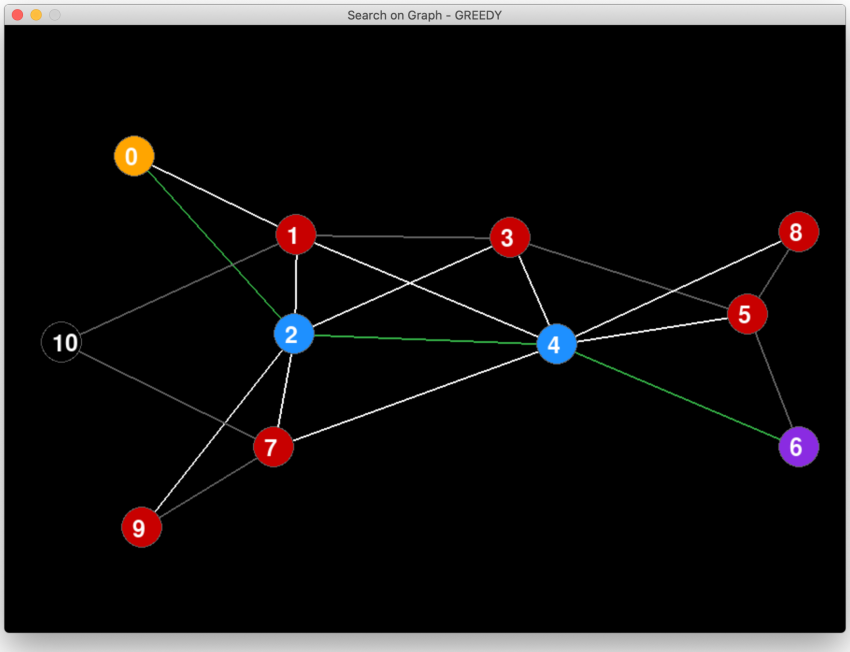

Figure 13: Kết quả tesecase1 của thuật toán Greedy

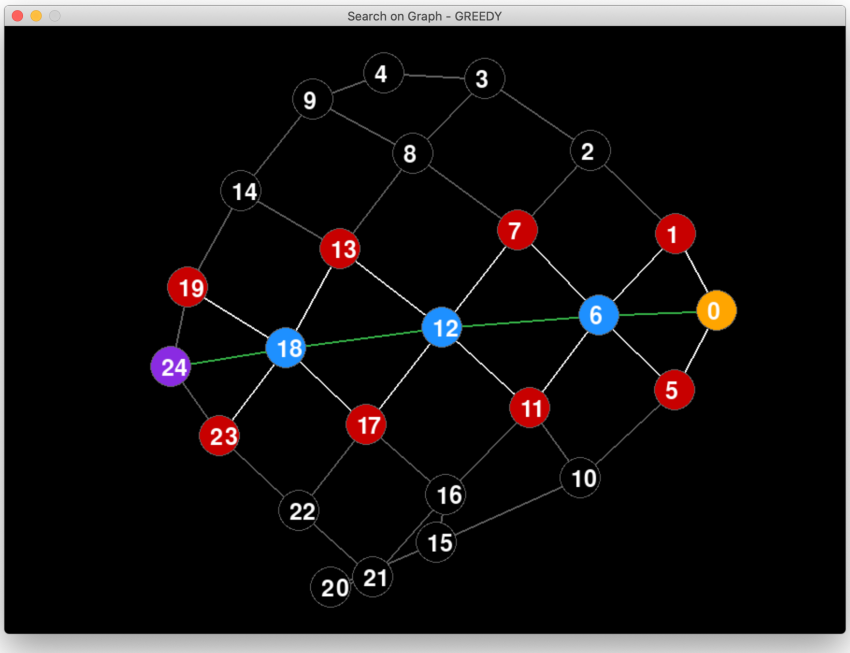

Figure 14: Kết quả tesecase2 của thuật toán Greedy

6 Tài liệu tham khảo

1. https://vi.wikipedia.org/wiki/Thuật_toán_Dijkstra
2. https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm
3. https://www.geeksforgeeks.org/a-search-algorithm/
4. https://www.geeksforgeeks.org/search-algorithms-in-ai/
5. https://www.massey.ac.nz/~mjjohnso/notes/59302/l04.html