BÁO CÁO BÀI TẬP NHÓM CUỐI KỲ: BÀI TOÁN VẬN CHUYỂN GÓI HÀNG THEO THỜI GIAN THỰC VỚI ĐA TÁC TỬ

Cao Đặng Quốc Vương
MSV: 22022601
Email: 22022601@vnu.edu.vn

Hồ Minh Hoàng
MSV: 22022567
Email: 22022567@vnu.edu.vn

Mã nguồn: https://github.com/vuong020304/marl-delivery

***Tóm tắt nội dung*—Báo cáo này trình bày bài toán vận chuyển gói hàng theo thời gian thực với đa tác tử trên lưới kích thước $N \times N$ có vật cản. Nhiều robot phối hợp để nhận và giao các gói hàng đúng hạn, tối ưu tổng phần thưởng nhận được. Các giải thuật tiếp cận như BFS tham lam, A* . Kết quả thực nghiệm cho thấy sự khác biệt về hiệu quả giữa các phương pháp.**

I. Giới thiệu

Bài toán được định nghĩa như sau:

Cho một bảng A có kích thước $N \times N$, trong đó giá trị $A_{i,j} = 0$ ứng với ô trống và $A_{i,j} = 1$ ứng với ô có vật cản (không thể di chuyển vào), với i là dòng, j là cột, $1 \leq i, j \leq N$.

Tại thời điểm $t = 0$, có C robot làm nhiệm vụ vận chuyển hàng hoá với toạ độ (x_i, y_i) với $0 \leq i < C$. Mỗi robot đứng ở một ô không phải vật cản và không có hai robot nào đứng cùng một ô.

Tại mỗi thời điểm $0 < t < T$ (T là thời gian lớn nhất), mỗi robot có thể thực hiện một trong 5 hành động di chuyển: S, L, R, U, D và nhặt gói hàng (1) hoặc trả gói hàng (2), hoặc không làm gì (0). Cụ thể:

- S: Đứng tại chỗ.
- L: Di chuyển sang trái.
- R: Di chuyển sang phải.
- U: Di chuyển lên trên.
- D: Di chuyển xuống dưới.
- 1: Nhặt hàng, nếu ô đang đứng sau khi di chuyển có gói hàng.
- 0: Không làm gì.
- 2: Thả gói hàng, nếu ô đang đứng sau khi di chuyển là ô mục tiêu của gói hàng.

Tại mỗi thời điểm $0 \leq t < T$, có một số yêu cầu chuyển các gói hàng sẽ xuất hiện. Mỗi gói hàng sẽ có thông tin: $g_i = \langle s_{1i}, s_{2i}, e_{1i}, e_{2i}, e_{3i} \rangle$ trong đó với $i \ll G$:

- s_{1i}, s_{2i}: Toạ độ hàng và cột của điểm bắt đầu của gói hàng thứ i.
- e_{1i}, e_{2i}, e_{3i}: Toạ độ hàng và cột của điểm kết thúc, và hạn giao hàng. Quá thời hạn này sẽ bị tính là quá hạn.

Mỗi gói hàng được vận chuyển đúng hạn sẽ được tính tiền công vận chuyển là $r \geq 0$, quá hạn là $r' \geq 0$. Trong thiết lập này, giá trị phần thưởng được chọn cố định là $r = 10$ và $r' = 1$.

Mỗi robot có một chi phí di chuyển cố định là $r_m = -0.01$. Chi phí này chỉ tính khi robot di chuyển (thực hiện hành động L, R, U, D).

II. Giải thuật A* Tối ưu

A. Ý tưởng và vai trò của A*

Trong bài toán vận chuyển gói hàng đa tác tử, mỗi robot cần liên tục xác định đường đi tối ưu từ vị trí hiện tại đến điểm nhặt hàng, và từ điểm nhặt đến điểm giao hàng, đồng thời phải tránh vật cản và các robot khác. Để giải quyết bài toán này, nhóm lựa chọn thuật toán A* làm nền tảng cho việc tìm đường.

A* là một thuật toán tìm kiếm có hướng dẫn (informed search), kết hợp giữa chi phí thực tế đã đi $g(n)$ và chi phí ước lượng còn lại đến đích $h(n)$, giúp tìm ra đường đi ngắn nhất một cách hiệu quả. Hàm đánh giá của A* tại mỗi node được xác định như sau:

$$f(n) = g(n) + h(n)$$

Trong đó:

- $g(n)$: tổng chi phí từ điểm xuất phát đến node hiện tại n.
- $h(n)$: ước lượng chi phí từ node n đến đích (heuristic).

Trong môi trường lưới (grid), nhóm sử dụng Manhattan distance làm heuristic vì nó phù hợp với di chuyển 4 hướng (trên, dưới, trái, phải) và cho kết quả tốt nhất so với các loại heuristic khác như Euclidean hay Diagonal.

B. Quy trình hoạt động của giải thuật

1. Tìm đường đi cho robot

Mỗi khi robot cần di chuyển đến một vị trí mới (điểm nhặt hoặc điểm giao hàng), agent sẽ gọi hàm `a_star_search(start, goal)` để xác định chuỗi các bước di chuyển tối ưu. Quy trình cụ thể như sau:

1) Kiểm tra cache. Nếu đã từng tính đường đi từ start đến goal, trả về luôn để tiết kiệm thời gian.
2) Nếu start hoặc goal không hợp lệ (ra ngoài bản đồ hoặc là vật cản), trả về rỗng.

3) Khởi tạo open_list (priority queue) với node xuất phát, closed_set rỗng.
4) Lặp cho đến khi open_list rỗng:
   - Lấy node có giá trị f nhỏ nhất ra khỏi open_list.
   - Nếu node này là đích, truy vết lại đường đi và trả về.
   - Nếu node đã xét rồi thì bỏ qua.
   - Mở rộng các node lân cận hợp lệ (không ra ngoài bản đồ, không vào vật cản), tính toán lại g, h, f và đưa vào open_list nếu tốt hơn.
5) Nếu không tìm thấy đường đi, trả về rỗng.

2. Phân công gói hàng cho robot

Khi robot không mang hàng, agent sẽ tìm gói hàng phù hợp nhất để phân công. Việc này dựa trên:

- Độ dài đường đi từ robot đến điểm nhặt hàng (tính bằng A*).
- Độ dài đường đi từ điểm nhặt đến điểm giao hàng (cũng dùng A*).
- Deadline của gói hàng: các gói sắp hết hạn sẽ được ưu tiên hơn.
- Lợi nhuận dự kiến (reward trừ chi phí di chuyển).

Agent xây dựng một ma trận chi phí cho tất cả các cặp robot–gói hàng, sau đó phân công theo nguyên tắc greedy: cặp nào có chi phí thấp nhất sẽ được ghép trước.

3. Sinh hành động cho robot

Sau khi xác định mục tiêu, agent sẽ quyết định hành động cho từng robot:

- Nếu robot đang mang hàng và đã đến điểm giao, thực hiện thả hàng.
- Nếu robot đang mang hàng nhưng chưa đến đích, dùng A* để tìm bước tiếp theo.
- Nếu robot không mang hàng, tìm gói hàng phù hợp, di chuyển đến điểm nhặt, và thực hiện nhặt hàng khi đến nơi.
- Nếu không còn gói hàng nào, robot sẽ đứng yên.

4. Xử lý xung đột và deadlock

Trong môi trường nhiều robot, có thể xảy ra va chạm hoặc deadlock (nhiều robot muốn vào cùng một ô hoặc tạo thành chu trình di chuyển). Agent xử lý như sau:

- Phát hiện các chu trình di chuyển (cycle) giữa các robot. Nếu phát hiện, robot có ID nhỏ nhất trong chu trình sẽ đứng yên để phá deadlock.
- Nếu nhiều robot cùng hướng đến một vị trí, robot có ID lớn hơn sẽ đứng yên.
- Nếu robot bị kẹt (đứng yên quá lâu), agent sẽ cho robot thử di chuyển ngẫu nhiên để thoát khỏi trạng thái này.

5. Tối ưu hiệu suất

- **Cache đường đi**: Để tránh tính toán lặp lại, các đường đi đã tìm được sẽ được lưu lại trong cache.
- **Ưu tiên deadline**: Khi phân công gói hàng, agent tăng độ ưu tiên cho các gói hàng sắp hết hạn để tối đa hóa reward.

C. Đánh giá và nhận xét

Giải thuật A* kết hợp với phân công heuristic giúp agent hoạt động hiệu quả trong môi trường nhiều robot, nhiều gói hàng, bản đồ phức tạp. Việc sử dụng heuristic Manhattan phù hợp với bản đồ lưới, đảm bảo tốc độ và chất lượng giải pháp. Cơ chế xử lý xung đột và robot bị kẹt giúp hệ thống vận hành ổn định, tránh deadlock.

Tuy nhiên, giải thuật này tối ưu cục bộ, chưa xét đến tối ưu toàn cục như Min-Cost Max-Flow. Trong các trường hợp bản đồ rất lớn hoặc số lượng robot/gói hàng nhiều, có thể cần bổ sung các kỹ thuật tối ưu hóa nâng cao hơn.

III. Giải thuật Greedy Agents Optimal

A. Ý tưởng giải thuật

Thuật toán Greedy Agent Optimal là một thuật toán tham lam cải tiến cho bài toán điều phối đa robot vận chuyển gói hàng. Thuật toán sử dụng chiến lược "tham lam" - chọn gói hàng "tốt nhất" tại mỗi thời điểm, nhưng kết hợp nhiều cơ chế thông minh để xử lý các tình huống phức tạp như deadlock, livelock và va chạm.

B. Quy trình hoạt động

Khởi tạo và cập nhật trạng thái

Tạo danh sách robot với vị trí (x,y) và trạng thái mang gói; khởi tạo danh sách gói hàng với điểm lấy, điểm giao và thời hạn

- Theo dõi lịch sử di chuyển và phát hiện mẫu lặp lại
- Cập nhật thời gian đứng yên và xác định robot bị kẹt
- Phân tích bản đồ để xác định vùng có khả năng deadlock

Phân công và thực hiện nhiệm vụ

Trước khi quyết định di chuyển, thuật toán kiểm tra tình trạng kẹt tổng thể. Nếu hơn 60

- Robot đã mang gói: Sử dụng BFS để tìm đường đến điểm giao hàng
- Robot đã nhận nhiệm vụ nhưng chưa đến gói: Tìm đường đến điểm lấy hàng
- Robot chưa có nhiệm vụ: Tính toán điểm ưu tiên cho mỗi gói hàng còn trống dựa trên khoảng cách đến điểm lấy, khoảng cách từ điểm lấy đến điểm giao, và lịch sử thất bại, sau đó chọn gói có điểm thấp nhất

Thuật toán BFS được sử dụng để tìm đường đi ngắn nhất, khởi tạo từ điểm đích và lan truyền theo 4 hướng (lên, xuống, trái, phải) cho đến khi tìm thấy vị trí hiện tại của robot, sau đó trả về hành động tiếp theo và khoảng cách.

Xử lý deadlock, livelock và va chạm

Thuật toán đặc biệt chú trọng vào việc xử lý các tình huống phức tạp

1) **Phát hiện deadlock:** Nếu robot đứng yên quá lâu (`idle_time > 5`), thuật toán xác định robot đang bị kẹt. Nếu robot kẹt quá lâu với cùng một gói hàng (`idle_time > 10`) sau 2-3 lần thử không thành công, robot sẽ từ bỏ và tìm gói khác.
2) **Phát hiện livelock:** Thuật toán kiểm tra chuỗi di chuyển gần đây để phát hiện mẫu như LRLR, UDUD hoặc phát hiện robot chỉ di chuyển qua 2-3 vị trí trong 6 bước

gần đây. Khi phát hiện, thuật toán sẽ tạo di chuyển ngẫu nhiên ưu tiên các vị trí ít đi qua và tránh vùng có khả năng deadlock.

3) **Xử lý robot r3:** Thuật toán có cơ chế đặc biệt cho robot r3 (`index 2`) - phát hiện khi robot chỉ di chuyển theo trái-phải và buộc nó thử di chuyển theo hướng lên-xuống, giúp phá vỡ mẫu lặp không hiệu quả.
4) **Tránh va chạm:** Thuật toán dự đoán vị trí tiếp theo của mỗi robot dựa trên hành động và phát hiện các xung đột tiềm ẩn. Khi phát hiện xung đột, thuật toán sẽ tìm hướng di chuyển thay thế, ưu tiên robot đang thực hiện nhặt/thả gói. Các robot đứng yên quá lâu được buộc di chuyển theo hướng ngược với lần di chuyển trước.

Hoàn thiện và trả về hành động

Mục tiêu của giai đoạn này là hoàn tất danh sách hành động của tất cả các robot, đồng thời chuẩn bị thông tin cho bước mô phỏng tiếp theo. Quy trình cụ thể bao gồm các bước sau:

1) **Đảm bảo mọi robot đều có hành động:**
   - *Hướng di chuyển* được mã hóa theo 5 ký hiệu: `U` (lên), `D` (xuống), `L` (trái), `R` (phải), `S` (đứng yên).
   - *Thao tác với gói hàng* gồm 3 hành động: `0` (không làm gì), `1` (nhặt gói), `2` (thả gói).
2) **Kiểm tra xung đột vị trí:** Hệ thống đảm bảo không có hai robot nào cùng di chuyển đến một ô lưới trong cùng một bước, tránh xảy ra va chạm.
3) **In thông tin trạng thái (tùy chọn debug):**
   - Tổng số lượng robot trong hệ thống.
   - Mục tiêu hiện tại hoặc trạng thái nhiệm vụ của mỗi robot.
4) **Trả về danh sách hành động:** Danh sách hành động có dạng: `actions = [(move_1, action_1), (move_2, action_2), ..., (move_n, action_n)]` Trong đó `move_i` là hướng di chuyển, và `action_i` là thao tác thực hiện với gói hàng của robot thứ i.
5) **Chuẩn bị cho bước tiếp theo:** Hệ thống lưu trữ các thông tin cần thiết cho lần gọi `get_actions()` tiếp theo, như trạng thái robot, thời gian chờ, hay thông tin gói hàng đã nhận.

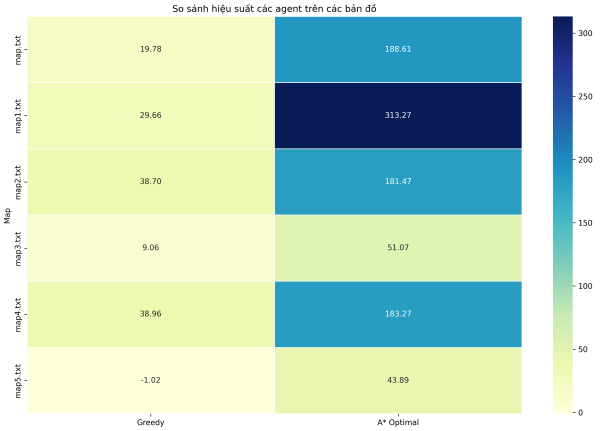
Thuật toán Greedy Optimal đạt được hiệu suất cao nhờ kết hợp giữa tối ưu hóa tham lam với các cơ chế thông minh để xử lý tình huống phức tạp, giúp cải thiện đáng kể hiệu quả vận chuyển so với các thuật toán tham lam thông thường.

IV. So sánh reward với phương pháp baseline

Để đánh giá hiệu quả của giải thuật A* Tối ưu, nhóm đã tiến hành thực nghiệm so sánh với phương pháp baseline là Greedy BFS trên 6 bản đồ khác nhau. Kết quả tổng phần thưởng (reward) thu được của từng phương pháp trên mỗi bản đồ được thể hiện trong Hình 1.

Nhận xét:

- Trên tất cả các bản đồ kiểm thử, giải thuật A* Tối ưu đều đạt tổng reward vượt trội so với Greedy BFS. Ví dụ, trên bản đồ `map1.txt`, reward của A* Tối ưu cao gấp nhiều lần so với Greedy BFS.



Hình 1. So sánh reward giữa Greedy BFS và A* Tối ưu trên các bản đồ kiểm thử

- Sự chênh lệch này đặc biệt rõ rệt ở các bản đồ có nhiều vật cản hoặc nhiều gói hàng, nơi mà việc tối ưu đường đi và phân công hợp lý đóng vai trò quan trọng.
- Ở một số bản đồ lớn hơn, reward của Optimal A* vẫn cao gấp nhiều lần so với Greedy BFS.
- Đáng chú ý, trên một số bản đồ, Greedy BFS thậm chí có reward rất thấp hoặc âm, trong khi A* Tối ưu vẫn đạt giá trị dương ổn định, cho thấy khả năng tối ưu hóa hiệu quả của giải thuật đề xuất.

Kết quả này khẳng định rằng giải thuật A* Tối ưu không chỉ giúp tối đa hóa phần thưởng nhận được mà còn ổn định và hiệu quả hơn hẳn so với phương pháp Greedy BFS, đặc biệt trong các môi trường phức tạp.

V. Kết quả thực nghiệm

Để đánh giá hiệu quả của giải thuật A* Tối ưu, nhóm đã tiến hành thực nghiệm trên 6 bản đồ kiểm thử với các cấu hình khác nhau về số lượng robot, gói hàng và vật cản. Kết quả so sánh tổng phần thưởng (reward) giữa giải thuật A* Tối ưu và phương pháp baseline Greedy BFS được trình bày trong Bảng I.

Bảng I
So sánh reward giữa Greedy BFS và A* Tối ưu trên các bản đồ kiểm thử

Bản đồ	Greedy BFS	A* Tối ưu	Greedy Optimal
map.txt	19.78	188.61	28.10
map1.txt	29.66	313.27	39.43
map2.txt	38.70	181.47	45.77
map3.txt	9.06	51.07	19.21
map4.txt	38.96	183.27	40.84
map5.txt	-1.02	43.89	8.32

Nhận xét:

- Trên tất cả các bản đồ, giải thuật A* Tối ưu đều đạt tổng reward cao hơn rõ rệt so với Greedy BFS và Greedy Optimal. Điều này cho thấy hiệu quả vượt trội của việc kết hợp tìm đường tối ưu với phân công động và ưu tiên deadline.

- Sự chênh lệch reward giữa hai phương pháp càng lớn khi bản đồ có nhiều vật cản hoặc số lượng gói hàng lớn, nhấn mạnh vai trò của tối ưu hóa đường đi và phân công hợp lý trong môi trường phức tạp.
- Ở một số bản đồ, Greedy BFS thậm chí có reward rất thấp hoặc âm, trong khi A* Tối ưu vẫn duy trì được reward dương ổn định.
- Kết quả thực nghiệm khẳng định rằng A* Tối ưu không chỉ giúp tối đa hóa phần thưởng mà còn đảm bảo tính ổn định và hiệu quả cho hệ thống robot đa tác tử.

Nhìn chung, giải thuật A* Tối ưu đã chứng minh được ưu thế rõ rệt so với phương pháp baseline, đặc biệt trong các tình huống thực tế có nhiều ràng buộc và yêu cầu tối ưu hóa phức tạp. Tuy nhiên Greedy Optimal cũng đã có những cải tiến đáng kể so với Greedy BFS

- Cơ chế xử lý deadlock/livelock: Greedy Optimal phát hiện và xử lý tình huống robot bị kẹt, trong khi Greedy Agent thông thường không có khả năng này
- Cơ chế tránh va chạm thông minh: Dự đoán và giải quyết xung đột giữa các robot, thay vì xử lý đơn giản hoặc không xử lý như Greedy Agent
- Khả năng từ bỏ nhiệm vụ: Có thể từ bỏ gói hàng khi robot bị kẹt quá lâu, giúp hệ thống không bị đình trệ

VI. Thảo luận

Kết quả thực nghiệm cho thấy giải thuật A* Tối ưu vượt trội so với phương pháp baseline Greedy BFS về tổng phần thưởng (reward) trên tất cả các bản đồ kiểm thử. Sự khác biệt này xuất phát từ một số yếu tố chính sau:

- **Tối ưu hóa đường đi:** A* Tối ưu sử dụng thuật toán A* với heuristic Manhattan để tìm đường đi ngắn nhất, giúp các robot di chuyển hiệu quả hơn, giảm chi phí di chuyển và tăng khả năng giao hàng đúng hạn. Trong khi đó, Greedy BFS chỉ xét đến khoảng cách gần nhất mà không tối ưu hóa toàn cục.
- **Phân công động và ưu tiên deadline:** A* Tối ưu và Greedy Optimal xây dựng ma trận chi phí và ưu tiên các gói hàng sắp hết hạn, giúp tối đa hóa reward và giảm số lượng hàng bị giao trễ. Greedy BFS không xét đến deadline nên dễ bỏ lỡ các gói hàng quan trọng.
- **Xử lý xung đột và deadlock:** A* Tối ưu có cơ chế phát hiện và xử lý va chạm, deadlock giữa các robot, đảm bảo hệ thống vận hành ổn định. Điều này giúp giảm thiểu các tình huống robot bị kẹt hoặc lãng phí bước di chuyển.
- **Ổn định và hiệu quả:** Trên các bản đồ phức tạp, nhiều vật cản hoặc số lượng gói hàng lớn, sự chênh lệch reward giữa hai phương pháp càng rõ rệt, khẳng định tính ổn định và hiệu quả của A* Tối ưu trong môi trường thực tế.

Tuy nhiên, giải thuật A* Tối ưu cũng có một số hạn chế:

- **Chi phí tính toán:** Việc sử dụng A* và phân công động làm tăng thời gian tính toán mỗi bước so với Greedy BFS và Greedy Optimal, đặc biệt khi số lượng robot và gói hàng lớn.
- **Tối ưu cục bộ:** Phân công hiện tại vẫn mang tính cục bộ, chưa xét đến tối ưu toàn cục cho toàn bộ hệ thống trong dài hạn.

Nhìn chung, A* Tối ưu là lựa chọn phù hợp cho các hệ thống robot đa tác tử cần tối ưu hóa reward và đảm bảo vận hành ổn định trong môi trường phức tạp. Trong tương lai, có thể nghiên cứu bổ sung các kỹ thuật tối ưu hóa toàn cục hoặc học sâu để nâng cao hơn nữa hiệu quả của hệ thống.

VII. Kết luận

Báo cáo đã trình bày và đánh giá giải thuật A* Tối ưu và Greedy Optimal cho bài toán vận chuyển gói hàng thời gian thực với nhiều robot trên môi trường lưới có vật cản. Thông qua việc kết hợp thuật toán tìm đường A* với phân công động và ưu tiên deadline, giải thuật đã chứng minh hiệu quả vượt trội so với phương pháp baseline Greedy BFS về tổng phần thưởng nhận được trên nhiều bản đồ kiểm thử khác nhau.

Giải thuật A* Tối ưu không chỉ giúp tối đa hóa reward mà còn đảm bảo tính ổn định, giảm thiểu va chạm và deadlock giữa các robot. Tuy nhiên, giải thuật vẫn còn một số hạn chế về chi phí tính toán và tối ưu cục bộ.

Trong tương lai, nhóm sẽ tiếp tục nghiên cứu các phương pháp tối ưu hóa toàn cục, ứng dụng học sâu hoặc các kỹ thuật phân phối để nâng cao hơn nữa hiệu quả và khả năng mở rộng của hệ thống robot đa tác tử trong các môi trường thực tế phức tạp.

Phụ lục

A. Tham số thực nghiệm

- Số lượng bản đồ kiểm thử: 6
- Số robot: 5
- Số gói hàng: 100
- Số bước tối đa: 1000
- Phần thưởng giao đúng hạn: $r = 10$
- Phần thưởng giao quá hạn: $r' = 1$
- Chi phí di chuyển mỗi bước: $r_m = -0.01$
- Heuristic A*: Manhattan distance

B. Định dạng dữ liệu đầu vào

- **Bản đồ:** Ma trận $N \times N$ với giá trị 0 (ô trống) và 1 (vật cản).
- **Robot:** Danh sách các vị trí (x_i, y_i) ban đầu, không trùng nhau và không nằm trên vật cản.
- **Gói hàng:** Mỗi gói có thông tin $(s_{1i}, s_{2i}, e_{1i}, e_{2i}, deadline)$.

C. Kết quả thực nghiệm chi tiết

Bảng II
So sánh reward giữa Greedy BFS và A* Tối ưu trên các bản đồ kiểm thử

Bản đồ	Greedy BFS	A* Tối ưu	Greedy Optimal
map.txt	19.78	188.61	28.10
map1.txt	29.66	313.27	39.43
map2.txt	38.70	181.47	45.77
map3.txt	9.06	51.07	19.21
map4.txt	38.96	183.27	40.84
map5.txt	-1.02	43.89	8.32

D. Phân chia công việc giữa các thành viên

Bảng III
BẢNG PHÂN CÔNG CÔNG VIỆC GIỮA CÁC THÀNH VIÊN

STT	**Công việc**	**Người phụ trách**
1	Cài đặt thuật toán A* Tối ưu	Cao Đặng Quốc Vương
2	Cài đặt thuật toán Greedy BFS Optimal	Hồ Minh Hoàng
3	Viết báo cáo, kiểm thử hệ thống	Cả hai thành viên