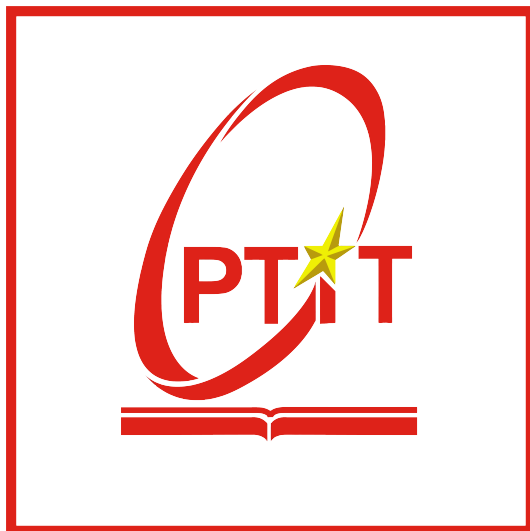


HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
CƠ SỞ TP HỒ CHÍ MINH



ĐỒ ÁN
CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

*Đề tài: Triển khai thuật toán PDFS offline trên đồ thị cây và
PDFS online trên đồ thị lưới, đánh giá hiệu năng*

Người hướng dẫn: TS. Nguyễn Minh Tuấn

Sinh viên thực hiện:

Bùi Phi Hùng	N23DCAT028
Phạm Anh Tuấn	N23DCAT074
Phan Thanh Dân	N23DCAT010
Trần Công Bảo	N23DCAT008
Lê Quốc Nhi	N23DCAT054

Mục lục

DANH MỤC CÁC KÝ HIỆU VÀ CHỮ VIẾT TẮT	2
1 CƠ SỞ LÝ THUYẾT	3
1.1 GIỚI THIỆU ĐỀ TÀI VỀ THUẬT TOÁN PDFS	3
1.2 CƠ SỞ THUẬT TOÁN	3
1.2.1 PDFS ONLINE	3
1.2.2 PDFS OFFLINE	8
1.3 MÃ GIẢ CỦA THUẬT TOÁN	14
1.3.1 PDFS OFFLINE	14
1.3.2 PDFS ONLINE	15
2 TRIỂN KHAI THUẬT TOÁN	19
2.1 TRIỂN KHAI PDFS OFFLINE	19
2.1.1 CODE	19
2.1.2 TEST	19
2.2 TRIỂN KHAI PDFS ONLINE	21
2.2.1 CODE	21
2.2.2 TEST	21
3 KIỂM TRA VÀ ĐÁNH GIÁ	23
3.1 KIỂM TRA TÍNH ĐÚNG ĐẮN	23
3.1.1 PDFS OFFLINE	23
3.1.2 PDFS ONLINE	26
3.2 ĐÁNH GIÁ THUẬT TOÁN	31
3.2.1 Ưu điểm của PDFS Offline và PDFS Online	31
3.2.2 Nhược điểm của PDFS Offline và PDFS Online	32
3.3 KẾT LUẬN	32
4 Mở rộng cho PDFS OFFLINE	34
4.1 Giới thiệu	34
4.2 Thiết lập và giải quyết vấn đề	34
4.2.1 Cơ sở lý thuyết	34
4.2.2 Quá trình thực hiện	35
4.3 Thuật toán	35
4.4 Triển khai code và đánh giá	37
4.4.1 Triển khai	37
4.4.2 Minh họa testcase và kiểm tra thủ công	37
4.4.3 Đánh giá và ứng dụng tương lai	40
DANH MỤC TÀI LIỆU THAM KHẢO	41

DANH MỤC CÁC KÝ HIỆU VÀ CHỮ VIẾT TẮT

Chữ viết tắt, ký hiệu	Ý nghĩa
DFS	Duyệt theo chiều sâu
PDFS (Piecemeal DFS)	Duyệt theo chiều sâu chia từng phần
PDFS Online	Duyệt PDFS không biết trước môi trường
PDFS Offline	Duyệt DFS biết trước môi trường
ECTC	Bài toán khám phá cây giới hạn năng lượng
B	Giới hạn năng lượng của tuyến đường

Chương 1

CƠ SỞ LÝ THUYẾT

1.1 GIỚI THIỆU ĐỀ TÀI VỀ THUẬT TOÁN PDFS

Thông thường thuật toán DFS chúng ta được biết đến là DFS với năng lượng vô hạn, nhưng thực tế hoàn toàn ngược lại, chúng bị giới hạn bởi một giá trị B , nên ta gọi là Piecemeal DFS (PDFS) hay là duyệt theo chiều sâu chia từng phần. Đặc biệt thuật toán PDFS này có thể duyệt trong môi trường chưa biết trước và cả môi trường biết trước được gọi là PDFS Online và PDFS Offline. Trong bài đề án này, chúng em sẽ trình bày chi tiết về hai thuật toán này, PDFS Offline trên đồ thị cây và PDFS Online trên đồ thị lưới. Quá trình nghiên cứu của bài đề án bao gồm:

1. Hiểu PDFS: duyệt chiều sâu giới hạn năng lượng (Offline, Online) [Chương 1].
2. Triển khai PDFS Offline (cây), PDFS Online (lưới) [Chương 2].
3. Kiểm tra thủ công và thuật toán chính xác [Mục 3.1].
4. Phân tích ưu nhược điểm, ứng dụng thực tế [Mục 3.2].
5. Tối ưu năng lượng dư trong quá trình duyệt [Chương 4].

1.2 CƠ SỞ THUẬT TOÁN

1.2.1 PDFS ONLINE

Định nghĩa

- PDFS ONLINE có nhiệm vụ tìm một đường đi hoặc một tập hợp các đường đi để phủ sóng tất cả các điểm trong một môi trường hoàn toàn chưa biết trước với ràng buộc năng lượng. Bài toán này có nhiều ứng dụng thực tế bao gồm quét tự động, hút bụi... Bởi vì chưa biết trước môi trường nên robot cần phải khám phá và tránh các chướng ngại vật chưa biết trong môi trường trong khi phủ sóng tất cả các điểm trong không gian trống bằng cách di chuyển với khoảng cách ngắn nhất có thể.
- Thuật toán đề xuất bao phủ một môi trường chưa biết (trực tuyến) thông qua phương pháp duyệt DFS được thiết kế riêng cho ngân sách năng lượng hạn chế B . Robot r thực hiện duyệt tìm kiếm theo chiều sâu trong khi xây dựng bản đồ cây của môi trường khi đang di chuyển. Nó quay trở lại điểm xuất phát để sạc pin khi độ dài đường đi của quá trình duyệt trở thành tối đa B . Sau khi pin được sạc đầy, r di chuyển đến ô nơi nó dừng ở quá trình DFS trước và tiếp tục duyệt P .

Công thức và định lý liên quan

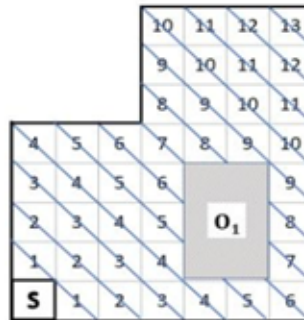
Phương pháp duyệt PDFS ONLINE

Môi trường khám phá

- Môi trường P là một đa giác phẳng chứa một trạm sạc S duy nhất bên trong nó. P có thể chứa các chướng ngại vật tĩnh, đa giác. Đặt robot r ban đầu tại trạm sạc S . r có kích thước $L \times L$ phù hợp với một ô lưới trong P . Robot r di chuyển theo đường thẳng trong P , tức là nó có thể di chuyển đến bất kỳ ô nào trong bốn ô lân cận (nếu ô không bị vật cản chiếm giữ) từ ô hiện tại của nó. Hơn nữa, giả định rằng ban đầu r không có bất kỳ thông tin nào về P , tức khám phá một môi trường P chưa biết. Để có thể phủ sóng tất cả các ô của P , ta giả định rằng P lớn bằng một hình tròn có bán kính $B/2$ với tâm tại S . Người ta cho rằng mức tiêu thụ năng lượng của robot tỷ lệ thuận với khoảng cách đã di chuyển, tức là ngân sách năng lượng của B cho phép robot di chuyển B đơn vị khoảng cách và mỗi lần đi của robot tiêu tốn 1 đơn vị năng lượng.

Xử lý môi trường chưa biết

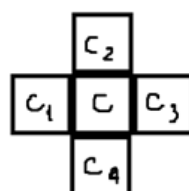
- Trong phần này, chúng ta sẽ tìm hiểu cách robot phân tích môi trường thành các ô lưới vuông và sau đó xây dựng bản đồ cây của môi trường T_P một cách nhanh chóng.
 - Phân tích môi trường P thành các ô vuông có kích thước $L \times L$, là kích thước của chính robot. Đường đồng mức khoảng cách bằng nhau là một đường đa tuyến trong đó các ô trên đó có cùng khoảng cách đến/từ điểm cơ sở S Hình 1.1. Các ô trên đường



Hình 1.1: Đường đồng mức khoảng cách

đồng mức có thể được sắp xếp từ bên này sang bên kia.

- Xây dựng bản đồ cây T_P : Ban đầu, robot r được đặt tại trạm sạc cố định S . Trong trường hợp này, cây chỉ có một nút S – trạm sạc của robot trong T_P .
- Robot r chọn ô trống đầu tiên c_1 theo thứ tự theo chiều kim đồng hồ của các ô lân cận của nó bắt đầu từ phía tây và kết thúc ở ô lân cận phía nam. Sau đó, r chèn nó vào T_P như một ô con của S nếu ô đó có thể tiếp cận được, nếu không r sẽ đến các ô lân cận khác cho đến khi tìm thấy ô đầu tiên có thể tiếp cận được.



Hình 1.2: Cách chọn ô

- Vì r đang xây dựng T_P trong khi khám phá P , nó sẽ di chuyển đến c_1 sau khi được đưa vào như một phần tử con trong T_P . Sau đó, robot r lại lặp lại quá trình xây dựng T_P từ ô c_1 hiện tại của nó.
- Có 1 vấn đề tiềm ẩn trong 1 số tình huống nhất định. Hãy xem xét môi trường được thể hiện trong hình dưới, trong đó đường ngang đi qua S đang vượt qua chướng ngại vật O_1 . Việc đánh số đường viền và xây dựng TP dựa trên số đường viền không hiệu quả vì các ô phải của O_1 không thể được ghé thăm theo thuật toán 1 vì nó yêu cầu r phải ghé thăm các ô theo thứ tự của số đường viền. Điều này là lí do số đường viền cho các ô đó nhỏ hơn số đường viền của các ô ở Bắc, Nam và Tây.

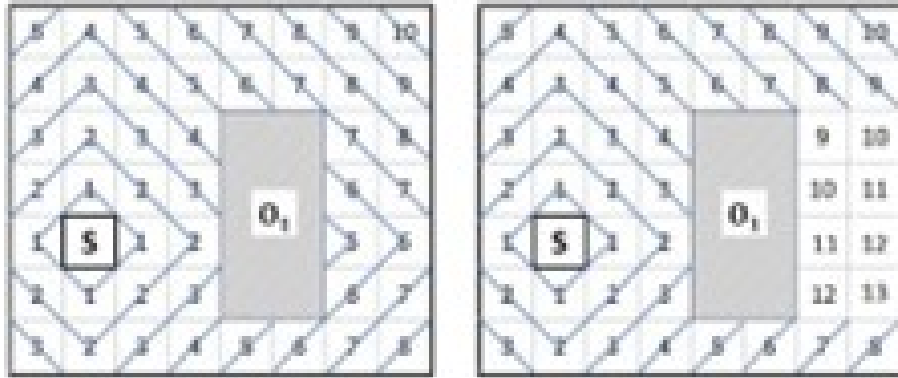


Fig. 2. An illustration of changes on contour numbers

Điều kiện khám phá

- Ban đầu robot ở trạm sạc S nằm bên trong P . Mục tiêu của thuật toán là tìm một tập hợp các tuyến đường $Q = \{Q_1, \dots, Q_k\}$ sao cho thỏa các điều kiện dưới đây:
 - Mỗi đường dẫn Q_i bắt đầu và kết thúc tại S .
 - : Mỗi đường dẫn Q_i có độ dài $l(Q_i) \leq B$.
 - : Các đường dẫn trong Q cùng nhau bao phủ môi trường P , tức là, $\bigcup_{i=1}^n Q_i = P$, và hai số liệu hiệu suất sau được tối ưu hóa:
 - * Số liệu hiệu suất 1: Số lượng tuyến đường trong Q , ký hiệu là $|Q|$ là tối thiểu.
 - * Số liệu hiệu suất 2: Tổng độ dài của các tuyến đường trong Q , ký hiệu là $l(Q) = \sum l(Q_i)$ tối thiểu.
- Định lý 1: Cho một môi trường đa giác phẳng P chưa biết có thể chứa chướng ngại vật và một robot r có kích thước $L \times L$ bao gồm các cảm biến phát hiện vị trí và chướng ngại vật ban đầu được đặt tại một trạm sạc S bên trong P với ngân sách năng lượng B , có một thuật toán giải đúng ONLINECPP và đảm bảo xấp xỉ **10 lần** cho cả hai số liệu hiệu suất so với thuật toán tối ưu có đầy đủ kiến thức về P .
- Định lý 2: Cho một môi trường đa giác phẳng chưa biết P có thể chứa chướng ngại vật với một robot r có ngân sách giới hạn là B ban đầu được đặt tại một trạm sạc S bên trong P , PDFS Online là để r đến thăm tất cả các ô có thể tiếp cận của P thông qua một tập hợp các tuyến sao cho thỏa các điều kiện nêu ở phần trên.

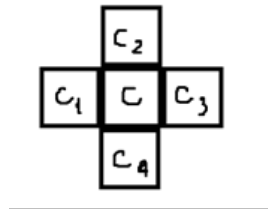
Mô tả thuật toán**• Ban đầu:**

- Robot được đặt tại S – trạm sạc và có giới hạn năng lượng là B .
- Bắt đầu với năng lượng hiện tại là 0 ($\text{curE} = 0$).

• Bước 1: Từ ô đang xét, xét ô lân cận ở 4 hướng: trái, lên, phải, xuống, nếu ô đó:

- Không phải là chướng ngại vật (giá trị khác 1).
- Trong ma trận.
- Chưa thăm.

Thêm các ô đó vào 4 stack tương ứng. Ví dụ:



Hình 1.3: Cách chọn ô


- Ta có 4 stack: Ban đầu **stL** (stack chứa các ô bên trái điểm đang xét), **stU** (các ô bên trên), **stR** (các ô bên phải), **stD** (các ô bên dưới) rỗng.
- Điểm đang xét là c .
- Điểm c_1 là điểm bên trái c nên thêm vào **stL**.
- Điểm c_2 là điểm bên trên c nên thêm vào **stU**.
- Điểm c_3 là điểm bên phải c nên thêm vào **stR**.
- Điểm c_4 là điểm bên dưới c nên thêm vào **stD**.

Lưu ý duyệt các stack theo thứ tự Tây \rightarrow Bắc \rightarrow Đông \rightarrow Nam, tức là ưu tiên duyệt điểm bên trái nhất của điểm đang xét (c). Cụ thể: Sau khi thêm các ô lân cận, ta được các stack: $\text{stL} = c_1$, $\text{stU} = c_2$, $\text{stR} = c_3$, $\text{stD} = c_4$. Khi duyệt, chúng ta phải ưu tiên lấy điểm từ **stL** \rightarrow **stU** \rightarrow **stR** \rightarrow **stD**.

• Bước 2: Khi đi đến một ô mới

- Nếu bắt đầu tuyến mới:
 - * Đặt năng lượng hiện tại (curE) bằng số bước từ S – xuất phát đến ô đang xét.
 - * Lưu lại hành trình từ S đến ô này (dựa trên $\text{par}[] []$ - cha của mỗi ô trong đường đi).
- Nếu đang trong route:
 - * Tính số năng lượng cần để đi thêm bước tiếp theo (**stepE**).
 - * $\text{stepE} = d[\text{px}][\text{py}] - d[\text{par}[\text{x}][\text{y}].\text{first}][\text{par}[\text{x}][\text{y}].\text{second}]$
 - $d[\text{px}][\text{py}]$: là khoảng cách từ S đến điểm trước đó.
 - $d[\text{par}[\text{x}][\text{y}].\text{first}][\text{par}[\text{x}][\text{y}].\text{second}]$: là khoảng cách từ S đến điểm cha của điểm đang xét.
 - * Ý nghĩa: Tính khoảng cách từ điểm đang xét đến ô mới.

- * Kiểm tra xem năng lượng đã dùng ($curE$) + năng lượng cần thêm ($stepE$) + khoảng cách quay về ($d[x][y]$) có bằng năng lượng giới hạn (E) hay không:
 - Nếu $= E \rightarrow$ dừng lại, đánh dấu kết thúc route.
 - Nếu $> E \rightarrow$ kết thúc route cũ, bắt đầu tuyến mới tại ô hiện tại.
 - Nếu $< E \rightarrow$ tiếp tục thêm ô hiện tại vào hành trình.

	0	1
0	3	4
1	2	
2	1	2
3	5	1

Ví dụ:

- Giả sử Route của robot đi từ $s \rightarrow (2, 0) \rightarrow (1, 0) \rightarrow (0, 0) \rightarrow (0, 1) \rightarrow (2, 1)$
- Khi robot ở tại $(0, 1)$ có năng lượng hiện tại là 4
- Ô tiếp theo của nó là $(2, 1)$ nên robot sẽ di chuyển từ $(0, 1) \rightarrow (0, 0) \rightarrow (1, 0) \rightarrow (2, 0) \rightarrow (2, 1)$
- Vậy tổng năng lượng của tuyến là: 8
- \Rightarrow **Vậy cách tính năng lượng cần thêm như thế nào:**
 - Ta có: $par[2][1] = [2][0]$ (trong đường đi ngắn nhất từ s đến $[2][1]$)
Điểm đang xét là $[2][1]$
Điểm xét trước đó là $[0][1]$
 - Theo công thức: $stepE = |d[0][1] - d[2][0]| + 1 = |4 - 1| + 1 = 4$
 - Vậy cần 4 năng lượng để đi từ $[0][1] \rightarrow [2][0]$
- * Đánh dấu ô đang xét đã được thăm (để không quay lại).
- * Tiếp tục khám phá các ô kề của ô đang xét, và đưa chúng vào stack tương ứng.
- * Quay lại xét các ô tiếp theo trong các stack.

• Kết quả:

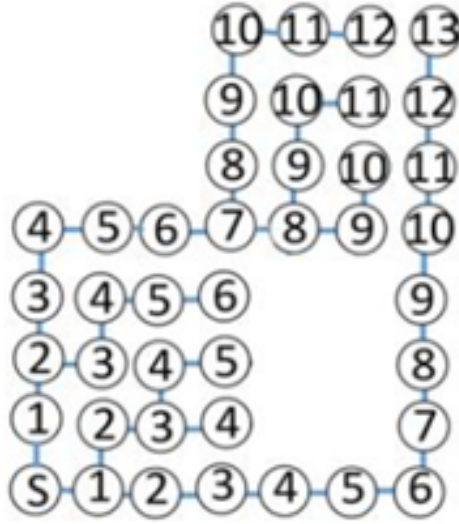
- In ra số đường đi đã tìm được và tổng năng lượng của từng đường.

B = 12		(0, 0)	(0, 1)	(0, 2)							
L = 1		(1, 0)	(1, 1)	(1, 2)							
S đặt tại (0, 0)		(2, 0)	(2, 1)	(2, 2)							
Chương ngại vật: (1, 1)											
Bước	Điểm xét	stL (Stack)	stU (Stack)	stR (Stack)	stD (Stack)	nodeNext (lấy từ Stack theo thứ tự)	Ql (từ S đến điểm cuối cùng rồi quay lại S)	curE	Dã thăm	Cập nhật đường đi tốt nhất từ S đến điểm đang xét	par[]
0 (Khởi tạo)	S = (0, 0)	{S}	{}	{}	{}	S	{S}	0	S	$d[0][0] = 0$	par[0][0] không xác định
1	S - (0, 0)	{}	{}	{(0, 1)}	{(1, 0)}	(0, 1)	{S; (0, 1)}	1	Thêm (0, 1)	$d[0][0] = 0$	par[0][0] = 1
2	(0, 1)	{}	{}	{(0, 2)}	{(1, 0)}	(0, 2)	{S; (0, 1); (0, 2)}	2	Thêm (0, 2)	$d[0][1] = 1$	par[0][1] = [0][0]
3	(0, 2)	{}	{}	{}	{(1, 0); (1, 2)}	(1, 2)	{S; (0, 1); (0, 2); (1, 2)}	3	Thêm (1, 2)	$d[0][2] = 2$	par[0][2] = [0][1]
4	(1, 2)	{}	{}	{}	{(1, 0); (2, 2)}	(2, 2)	{S; (0, 1); (0, 2); (1, 2); (2, 2)}	4	Thêm (2, 2)	$d[1][2] = 3$	par[1][2] = [0][2]
5	(2, 2)	{(2, 1)}	{}	{}	{(1, 0)}	(2, 1)	{S; (0, 1); (0, 2); (1, 2); (2, 2); (2, 1)}	5	Thêm (2, 1)	$d[2][2] = 4$	par[2][2] = [1][2]
6	(2, 1)	{(2, 0)}	{}	{}	{(1, 0)}	(2, 0)	{S; (0, 1); (0, 2); (1, 2); (2, 2); (2, 1); (2, 0)}	6	Thêm (2, 0)	$d[2][1] = 5$	par[2][1] = [2][2]
7	(2, 0)	{}	{(1, 0)}	{}	{(1, 0)}	(1, 0)	Kết thúc tuyến 1, Bắt đầu tuyến 2	1	Thêm (1, 0)	$d[2][0] = 6$	par[2][0] = [2][1]
8	(1, 0)	{}	{}	{}	{}	S	{S; (1, 0)}	1	Thăm hết	$d[1][0] = 1$	par[1][0] = [0][0]
Vậy có 2 tuyến		Q1: {S; (0, 1); (0, 2); (1, 2); (2, 2); (2, 1); (2, 0)}									
		Q2: {S; (1, 0)}									

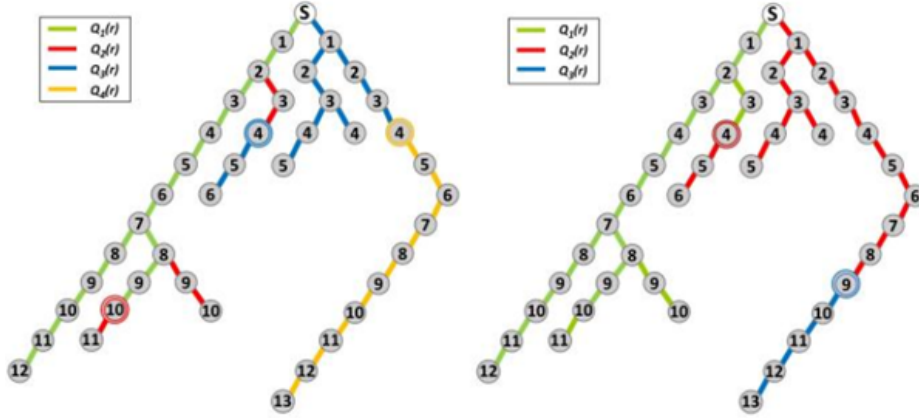
Hình 1.4: Số tuyến đường và tổng năng lượng

Minh họa thuật toán

- Xét trường hợp $B = 30$ và $B = 40$.
- Kết quả cây T_P .
- Kết quả các tuyến đường.



Hình 1.5: Cây T_P



Hình 1.6: Các tuyến đường

- Các điểm được đánh dấu bằng vòng tròn kép là điểm dừng của phép duyệt trước đó và quá trình duyệt tiếp theo bắt đầu từ nút này.
- Với $B = 40$ thì sẽ có ít tuyến đường hơn $B = 30$.

1.2.2 PDFS OFFLINE

Định nghĩa về ECTE và PDFS offline

ECTE: Là bài toán khám phá cây có trọng số bị ràng buộc bởi năng lượng $B > 1$, và chiều cao tối đa là $B/2$. Mục tiêu là tìm số nguyên k nhỏ nhất sao cho tồn tại một chiến lược thám hiểm B (B-exploration strategy) bao gồm k tuyến đường để bao phủ toàn bộ cây.

PDFS offline: Là thuật toán duyệt theo chiều sâu từng phần có giới hạn năng lượng B nhưng biết trước môi trường khám phá, cụ thể trong bài này môi trường khám phá là cây (tree). Mục tiêu của Bài toán là bao phủ toàn bộ cây T bằng cách sử dụng tuyến đường k nhỏ nhất với mỗi tuyến đường bắt đầu bắt đầu và kết thúc tại gốc và có độ dài không quá B , đã được đề cập trong tài liệu [1].

Công thức và định lý

Phương pháp duyệt DFS thông thường và PDFS offline:

DFS: [2] Cho G là đồ thị liên thông với tập đỉnh $\{v_1, v_2, \dots, v_n\}$

- 1 Lấy một đỉnh bất kỳ trong đồ thì đưa vào ngăn xếp.
- 2 Lấy top value của ngăn xếp để duyệt và thêm vào visited list.
- 3 Tạo một list bao gồm các đỉnh liền kề của đỉnh đang xét, thêm những đỉnh không có trong visited list vào ngăn xếp.
- 4 Tiếp tục lặp lại bước 2 và bước 3 đến khi ngăn xếp rỗng.

PDFS: [1] $\text{PDFS}(T) = (R_1, R_2, \dots, R_k)$ dựa trên $R_{DFS} = (v_0, v_1, \dots, v_l)$, lưu ý rằng là R_{DFS} có chứa cả các đỉnh quay lui. Dưới đây là các bước duyệt PDFS:

Bước 1. Mỗi tuyến đường R_i của PDFS đều phải kết thúc tại gốc (v_{j_0}, v_0, r) . Độ dài của mỗi tuyến đường R_i trong PDFS đều không thể vượt qua $B/2$. Tức là $l(R_i) \leq B$

Bước 2. Xây dựng các tuyến đường R_i :

- Với mỗi tuyến R_i , gọi $v_{j_{i-1}}$ là đỉnh xa nhất trên R_{DFS} mà tuyến đường trước đó (R_{i-1}) đã kết thúc phần khám phá mới là $R_1, v_{j_0} = r$.

- **Xác định điểm kết thúc cho tuyến R_i :**

- Tuyến R_i khởi đầu từ gốc đến $v_{j_{i-1}}$ (theo đường đi ngắn nhất).
- Từ $v_{j_{i-1}}$, tuyến R_i tiếp tục đi theo thứ tự của R_{DFS} (ví dụ: $v_{j_{i-1}}, v_{j_{i-1}+1}, \dots, v_p$) để đi đến những phần mới của cây.
- Việc mở rộng dừng tại đỉnh v_p sao cho điều kiện về năng lượng thỏa mãn:

$$d(r, v_{j_{i-1}}) + l(v_{j_{i-1}}, v_{j_{i-1}+1}, \dots, v_p) + d(v_p, r) \leq B \quad (1.1)$$

- Đỉnh v_p xa nhất mà thỏa mãn điều kiện trên ký hiệu là v_{j_i} , đây là điểm mà tuyến đường R_i ngừng việc khám phá.

Bước 3. Hình thành tuyến đường R_i :

- Tuyến đường được tạo thành bởi 3 phần:
 - Đường đi ngắn nhất từ gốc r đến $v_{j_{i-1}}$
 - Đoạn đường đi từ $v_{j_{i-1}}$ đến v_{j_i} , tức là đoạn $(v_{j_{i-1}}, v_{j_{i-1}+1}, \dots, v_p)$
 - Đoạn cuối là đoạn đi từ v_{j_i} đến gốc r

Bước 4. Nếu tất cả các cạnh và đỉnh của cây đều được đi qua thì dừng lại, ngược lại thì tăng i lên 1 đơn vị và quay lại bước 2.

Giới hạn về chi phí và số tuyến đường: [1]

Định lý 1: Cho cây T và khoảng cách xa nhất từ gốc đến lá trong T lớn nhất là $B/2$. Ta có $|PDFS(T)| \leq 12|R|$, với R là chiến lược khám phá mà với số tuyến là bé nhất

Tổng chi phí của khám phá $S = \{R_1, R_2, \dots, R_k\}$ được định nghĩa là tổng trọng số của tất cả các cạnh có trong S :

$$\xi(S) = \sum_{i=1}^k l(R_i)$$

Ta ký hiệu $COPT(T)$ là chiến lược khám phá tối ưu về chi phí, chiến lược khám phá này có chi phí nhỏ nhất trong các cách khám phá của PDFS.

Định lý 2 Cho cây T và $B/2$ lớn hơn hoặc bằng khoảng cách xa nhất từ gốc đến lá trong T , ta có: $\xi(PDFS(T)) \leq 12 \cdot \xi(COPT(T))$.

Chứng minh định lý 1:

Nhận xét: Có T và B , $|R| \geq \lceil \xi(COPT(T)/B) \rceil$, với R là chiến lược khám phá với số tuyến bé nhất

Xét cây T và T' với tuyến đường $R_{DFS} = (v_0, v_1, \dots, v_l)$. Với các tuyến đường của $PDFS(T)$ là $R = (R_1, R_2, \dots, R_k)$. Khi thực hiện $PDFS(T)$ trên cây T , các tuyến đường R nếu muốn đi tới thêm 1 đỉnh nữa thì sẽ vượt qua giới hạn mức năng lượng B , tức là:

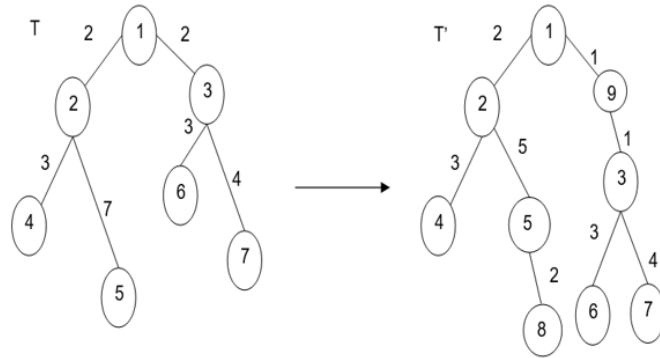
$$l(R_i) + 2\omega(v_{j_i}, v_{j_i+1}) > B$$

Ta chia nhỏ cạnh (v_{j_i}, v_{j_i+1}) thành 2 cạnh (v_{j_i}, v_x) và (v_x, v_{j_i+1}) , với trọng số:

$$\omega(v_{j_i}, v_x) = \frac{B - l(R_i)}{2}, \quad \omega(v_x, v_{j_i+1}) = \omega(v_{j_i}, v_{j_i+1}) - \frac{B - l(R_i)}{2}$$

Biến đổi cây gốc T thành cây T' để đảm bảo các tuyến PDFS có độ dài chính xác B (trừ tuyến cuối).

Theo hình vẽ ta thấy chi phí tối ưu của T và T' là như nhau, $\xi(COPT(T')) = \xi(COPT(T))$. Điều này có nghĩa là chiến lược tối ưu về chi phí sẽ không thay đổi giá trị giữa cây T và cây T' .



Hình 1.7: Hình minh họa chi phí tối ưu không thay đổi giữa T và T' với $B = 20$

Ta lại có:

$$|PDFS(T')| = \frac{\xi(PDFS(T'))}{B}$$

\Rightarrow Số lượng tuyến đường trên cây T và T' là như nhau $|PDFS(T)| = |PDFS(T')|$.

Chứng minh định lý 2:

Một cây T và có chiều dài L nếu cây đó thỏa mãn điều kiện $B > 1$ và chiều dài của cây phải nhỏ hơn hoặc bằng $B/2$. Khi đó số lượng tuyến đường trong $PDFS(T)$ là không quá 12 lần số lượng tuyến đường tối thiểu ($COPT$). Định lý 1: $|PDFS(T)| \leq 12 \cdot |R|$ Định lý chi phí của $PDFS$ không vượt quá 12 lần chi phí của chiến lược tối ưu:

$$\xi(PDFS(T)) \leq 12 \cdot \xi(COPT(T))$$

Từ các công thức trên ta có:

$$|PDFS(T')| = \frac{\xi(PDFS(T'))}{B} \leq \frac{12\xi(COPT(T'))}{B} \leq 12 \frac{\xi(COPT(T'))}{B}$$

Do đó:

$$|PDFS(T')| \leq 12 \frac{\xi(COPT(T'))}{B} = 12 \frac{\xi(COPT(T))}{B}$$

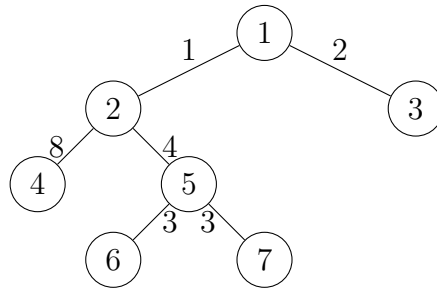
Ta lại thấy $|R| \geq \frac{\xi(COPT(T))}{B}$,

$$\Rightarrow 12 \cdot |R| \geq 12 \frac{\xi(COPT(T))}{B}$$

Kết hợp tất cả điều kiện trên định lý 1:

$$|PDFS(T')| = |PDFS(T)| \leq 12 \cdot |R| \quad (\text{đúng})$$

Trong đó R là con đường tốt nhất mà bị ràng buộc về năng lượng.



Hình 1.8: Cây đồ thị ví dụ với $B = 20$

Mô tả thuật toán trên cây hình 1.2 :

Thuật toán với $B = 20$:

- Duyệt cây theo DFS và đánh dấu các đỉnh:

- Khi duyệt bằng DFS thì thứ tự duyệt như sau:

$$R_{DFS} = (v_0, v_1, \dots, v_{12}) = (1, 2, 4, 2, 5, 6, 5, 7, 5, 2, 1, 3, 1)$$

- Ta đánh chỉ số các đỉnh như sau:

$$\begin{aligned} v_0 &= 1, & v_1 &= 2, & v_2 &= 4, & v_3 &= 2, & v_4 &= 5, & v_5 &= 6, \\ v_6 &= 5, & v_7 &= 7, & v_8 &= 5, & v_9 &= 2, & v_{10} &= 1, & v_{11} &= 3, & v_{12} &= 1 \end{aligned}$$

Thực hiện phép duyệt tuyến R_1 :

Bước 1: Tuyến đầu tiên nên $v_{j_0} = v_0 = 1$

Bước 2: Ta cần tìm v_p xa nhất thoả mãn điều kiện:

$$d(1, v_{j_0}) + l(v_{j_0}, v_{j_0+1}, \dots, v_p) + d(v_p, 1) \leq B \Rightarrow d(1, v_{j_0}) = d(1, 1) = 0$$

- Xét $v_p = v_0 = 1$, $d(1, v_{j_0}) + l(v_0, 1) + d(1, 1) = 0 + 0 + 0 \leq 20$ (thoả mãn)
- Xét $v_p = v_1 = 2$, $d(1, v_{j_0}) + l(1, 2) + d(2, 1) = 0 + 1 + 1 = 2 \leq 20$ (thoả mãn)
- Xét $v_p = v_2 = 4$, $d(1, v_{j_0}) + l(1, 2, 4) + d(4, 1) = 0 + 9 + 9 = 18 \leq 20$ (thoả mãn)
- Xét $v_p = v_3 = 2$, $d(1, v_{j_0}) + l(1, 2, 4, 2) + d(2, 1) = 0 + 17 + 1 = 18 \leq 20$ (thoả mãn)
- Xét $v_p = v_4 = 5$, $d(1, v_{j_0}) + l(1, 2, 4, 2, 5) + d(5, 1) = 0 + 21 + 5 = 26 > 20$ (không thoả mãn)

$\Rightarrow v_p$ xa nhất thoả mãn là $v_3 = 2$, tức $v_{j_1} = v_3$

Bước 3: Xây dựng tuyến R_1 :

- Đường đi từ r đến v_{j_0} : $d(1, 1) = 0$
- Đoạn đường từ v_{j_0} đến v_{j_1} là đoạn $(v_0, v_1, v_2, v_3) = (1, 2, 4, 2) = 17$
- Đoạn cuối là đoạn đi từ v_{j_1} đến gốc: $(v_3, 1) = (2, 1) = 1$

$$\Rightarrow \text{Tuyến } R_1 = (1, 2, 4, 2, 1)$$

Thực hiện phép duyệt tuyến R_2 :

Bước 1: Điểm bắt đầu khám phá $v_{j_1} = v_3 = 2$ (là v_3 trong R_{DFS})

Bước 2: Tìm v_p xa nhất có thể để thoả mãn điều kiện:

$$d(1, v_{j_1}) + l(v_{j_1}, v_{j_1+1}, \dots, v_p) + d(v_p, 1) \leq B \Rightarrow d(1, v_{j_1}) = d(1, 2) = 1$$

- Xét $v_p = v_3 = 2$, $d(1, v_{j_1}) + l(2, 2) + d(2, 1) = 1 + 0 + 1 = 2 \leq 20$ (thoả mãn)
- Xét $v_p = v_4 = 5$, $d(1, v_{j_1}) + l(2, 5) + d(5, 1) = 1 + 4 + 5 = 10 \leq 20$
- Xét $v_p = v_5 = 6$, $d(1, v_{j_1}) + l(2, 5, 6) + d(6, 1) = 1 + 7 + 8 = 16 \leq 20$
- Xét $v_p = v_6 = 5$, $d(1, v_{j_1}) + l(2, 5, 6, 5) + d(5, 1) = 1 + 10 + 5 = 16 \leq 20$
- Xét $v_p = v_7 = 7$, $d(1, v_{j_1}) + l(2, 5, 6, 5, 7) + d(7, 1) = 1 + 13 + 8 = 22 > 20$ (không thoả)

$\Rightarrow v_p = v_6 = 5$, $v_{j_2} = v_6$

Bước 3: Xây dựng tuyến R_2 :

- Đường đi từ r đến v_{j_1} : $d(1, 2) = 1$
- Đoạn đường từ v_{j_1} đến v_{j_2} : $(v_3, v_4, v_5, v_6) = (2, 5, 6, 5)$
- Đoạn cuối: $(v_6, 1) = (5, 1) = 5$

$$\Rightarrow \text{Tuyến } R_2 = (1, 2, 5, 6, 5, 2, 1)$$

Thực hiện phép duyệt tuyến R_3 :

Bước 1: Điểm bắt đầu khám phá $v_{j_2} = v_6 = 5$

Bước 2: Tìm v_p xa nhất để thoả mãn điều kiện:

$$d(1, v_{j_2}) + l(v_{j_2}, v_{j_2+1}, \dots, v_p) + d(v_p, 1) \leq B \Rightarrow d(1, v_{j_2}) = d(1, 5) = 1 + 4 = 5$$

- Xét $v_p = v_6 = 5$, ta có:

$$d(1, v_{j_2}) + l(5, 5) + d(5, 1) = 5 + 0 + 5 = 10 < 20 \quad (\text{thoả mãn})$$

- (Xét tương tự với $v_7, v_8, v_9, v_{10}, v_{11}, v_{12}$)

- Xét $v_p = v_{12} = 1$, ta có:

$$d(1, v_{j_2}) + l(5, 7, 5, 2, 1, 3, 1) + d(1, 1) = 5 + 15 + 0 = 20 \quad (\text{thoả mãn})$$

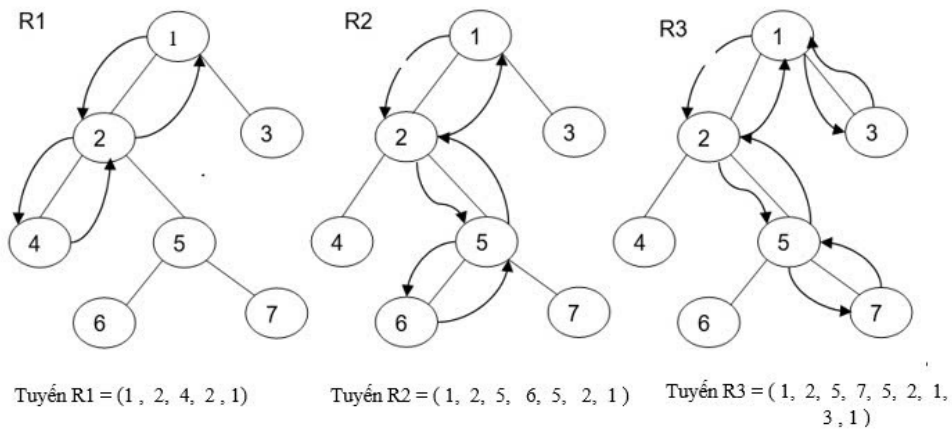
$\Rightarrow v_p = v_{12} = 1, v_{j_3} = v_{12}$

Bước 3: Xây dựng tuyến R_3 :

- Đường đi từ r đến v_{j_2} : $d(1, 5) = (1, 2, 5) = 5$
- Đoạn đường từ v_{j_2} đến v_{j_3} : $(v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}) = (5, 7, 5, 2, 1, 3, 1)$
- Đường đi từ v_{j_3} đến gốc : $d(v_{12}, 1) = (1, 1) = 0$

$$\Rightarrow \text{Tuyến } R_3 = (1, 2, 5, 7, 5, 2, 1, 3, 1)$$

Kết quả duyệt PDFS trên hình 1.1 :



Hình 1.9: Minh họa thuật toán PDFS trên cây đồ thị

Minh họa thuật toán

[PDFS Offline Auto](#)

1.3 MÃ GIẢ CỦA THUẬT TOÁN

1.3.1 PDFS OFFLINE

Algorithm 1 PDFS Offline Algorithm: Khởi tạo

```
procedure PDFS_offline
  path  $\leftarrow \emptyset$  {Nodes from root to current}
  currentRoute  $\leftarrow \emptyset$  {Current node IDs}
  routes  $\leftarrow \emptyset$  {All recorded routes}
  routeCosts  $\leftarrow \emptyset$  {Costs for each route}
  currentLength  $\leftarrow 0.0$  {Distance to current node}
  routeEnergy  $\leftarrow 0.0$  {Energy in current route}
  sumEnergy  $\leftarrow 0.0$  {Total energy used}
  routeCount  $\leftarrow 1$  {Route counter}
  fullDFS  $\leftarrow \emptyset$  {DFS traversal events}
  Add (root, 0.0) to path
  Add root.id to currentRoute, fullDFS
```

Algorithm 2 PDFS Offline Algorithm: Hàm DFS

```
function DFS(u)
for all (v, w)  $\in$  u.children do
  if not v.visited then
    if routeEnergy + w + (currentLength + w) > B then
      distToRoot  $\leftarrow$  currentLength
      routeEnergy  $\leftarrow$  routeEnergy + distToRoot
      for i  $\leftarrow$  path.size - 2 downto 0 do
        Add path[i].first.id to currentRoute
      end for
      Add currentRoute to routes
      Add routeEnergy to routeCosts
      routeCount  $\leftarrow$  routeCount + 1
      currentRoute  $\leftarrow$   $\emptyset$ 
      Add root.id to currentRoute
      routeEnergy  $\leftarrow$  0, newDist  $\leftarrow$  0
      for i  $\leftarrow$  1 to path.size - 1 do
        parentNode  $\leftarrow$  path[i - 1].first
        curNode  $\leftarrow$  path[i].first
        wt  $\leftarrow$  path[i].second
        newDist  $\leftarrow$  newDist + wt
        routeEnergy  $\leftarrow$  routeEnergy + wt
        sumEnergy  $\leftarrow$  sumEnergy + newDist
        Add curNode.id to currentRoute
      end for
      currentLength  $\leftarrow$  newDist
    else
      v.visited  $\leftarrow$  true
      Add v.id to fullDFS
      currentLength  $\leftarrow$  currentLength + w
      routeEnergy  $\leftarrow$  routeEnergy + w
      sumEnergy  $\leftarrow$  sumEnergy + currentLength
      Add v.id to currentRoute
      Add (v, w) to path
      DFS(v)
    end if
  end if
end for
```

1.3.2 PDFS ONLINE

Algorithm 3 PDFS Online Algorithm: Khởi tạo và Hàm run()

Đọc dữ liệu đầu vào:

- n, m, E : kích thước lưới, năng lượng tối đa
- $a[]$: ma trận lưới
- sx, sy : vị trí xuất phát ($a[i][j] == 2$)

Kiểm tra điều kiện năng lượng:

Điều kiện: $E \geq$ bán kính môi trường khám phá

Với tâm là vị trí xuất phát của robot

if $E < \text{bán kính môi trường khám phá}$ **then**

 Thông báo "Năng lượng không đủ để tìm đường đi!"

 Kết thúc chương trình

end if

(Xét các trường hợp $E \geq 4 \times \max(n, m)$)

Khởi tạo các biến:

- $d[]$: ma trận lưu khoảng cách từ xuất phát
- $vis[][]$: đánh dấu đã thăm
- $path[]$: lưu các route (mỗi route là vector Step)
- stL, stU, stR, stD : 4 stack cho 4 hướng (trái, lên, phải, xuống)
- $curE$: năng lượng hiện tại
- $newRoute$: đánh dấu bắt đầu route mới
- cnt : số route hiện tại
- px, py : vị trí trước đó

Bắt đầu:

- Đưa vị trí xuất phát (sx, sy) vào stL
- Gọi hàm $run()$

Hàm run():

while các stack không rỗng **do**

- Ưu tiên lấy từ stL , sau đó stU, stR, stD

if ô (x, y) lấy ra chưa thăm **then**

 Gọi $go(x, y)$

end if

end while

Algorithm 4 PDFS Online Algorithm: Hàm go() và upd()

Hàm go(x, y):

- Thêm bước hiện tại (x, y, d[x][y], curE) vào path[cnt]

if đang bắt đầu route mới (newRoute == false) **then**

- Đặt lại curE = d[x][y]

- newRoute = true

- Lưu lại route vừa đi (save(x, y))

else

- Tính năng lượng bước stepE (khoảng cách từ (px, py) đến (x, y))

if (x, y) không phải ô xuất phát **then****if** curE + stepE + d[x][y] == E **then**

- curE = curE + stepE

- Thêm bước (x, y, d[x][y], curE) vào path[cnt]

- newRoute = false

- Tăng cnt

else if curE + stepE + d[x][y] > E **then**

- Lưu lại route (save(x, y))

- Tăng cnt

- Đặt lại curE = d[x][y]

- newRoute = true

else

- curE = curE + stepE

- Thêm bước (x, y, d[x][y], curE) vào path[cnt]

end if**end if****end if**

- Đánh dấu vis[x][y] = 1

for mỗi hướng k (trái, lên, phải, xuống) **do**

- Tính tọa độ mới (nx, ny) từ (x, y) theo hướng k

if ô (nx, ny) hợp lệ (trong lưới, không phải vật cản) và chưa thăm (vis[nx][ny] == 0) **then**

- Đẩy (nx, ny) vào stack tương ứng (stL, stU, stR, stD)

- Gọi upd(x, y, nx, ny)

end if**end for**

- Cập nhật px = x, py = y

- Gọi lại run()

Hàm upd(x, y, nx, ny):**if** d[nx][ny] == 0 và d[x][y] + 1 ≤ E/2 **then**

- d[nx][ny] = d[x][y] + 1

- par[nx][ny] = (x, y)

else if d[nx][ny] > d[x][y] + 1 **then**

- d[nx][ny] = d[x][y] + 1

- par[nx][ny] = (x, y)

else if d[nx][ny] == d[x][y] + 1 **then**

- par[nx][ny] = (x, y)

end if

Algorithm 5 PDFS Online Algorithm: Hàm save() và In kết quả

Hàm save(x, y):

- Xóa path[cnt]
- Lăn ngược từ (x, y) về (sx, sy) theo par
- Lưu các bước vào path[cnt]
- Đảo ngược path[cnt] để đúng thứ tự

In kết quả:

- In số route: cnt
 - Với mỗi route trong path[]:
 - In các bước (x, y, d, e)
-

Chương 2

TRIỂN KHAI THUẬT TOÁN

2.1 TRIỂN KHAI PDFS OFFLINE

2.1.1 CODE

[Code PDFS Offline trên GitHub](#)

2.1.2 TEST

Input: Dòng 1: n B (số nút, năng lượng).

Dòng 2-n: cạnh có trọng số.

Output: Dòng 1: DFS quay lui.

Tiếp theo: các tuyến.

TestCase 1:

Input	Output
7 20	0 1 2 1 3 4 3 5 3 1 0 6 0
0 1 1	Route 1 (cost = 18): 0 1 2 1 0
0 6 2	Route 2 (cost = 16): 0 1 3 4 3 1 0
1 2 8	Route 3 (cost = 20): 0 1 3 5 3 1 0 6 0
1 3 4	
3 4 3	
3 5 3	

Bảng 2.1: TestCase 1 for PDFS Offline

TestCase 2:

Input	Output
11 24	0 1 2 7 2 8 2 1 4 1 0 3 5 9 5 10 5 3 6 3 0
0 1 3	Route 1 (cost = 16): 0 1 2 7 2 1 0
0 3 1	Route 2 (cost = 22): 0 1 2 8 2 1 0
1 2 1	Route 3 (cost = 18): 0 1 4 1 0 3 0
1 4 5	Route 4 (cost = 16): 0 3 5 9 5 3 0
2 7 4	Route 5 (cost = 24): 0 3 5 10 5 3 0
2 8 7	Route 6 (cost = 4): 0 3 6 3 0
3 5 6	
3 6 1	
5 9 1	
5 10 5	

Bảng 2.2: TestCase 2 for PDFS Offline

2.2 TRIỂN KHAI PDFS ONLINE

2.2.1 CODE

[Code PDFS Online trên GitHub](#)

2.2.2 TEST

Input: Dòng 1: hàng, cột, B.

Tiếp theo: 2 (xuất phát), 1 (vật cản), 0 (môi trường).

Output: Số tuyến và các tuyến.

TestCase 1	TestCase 2
8 8 64 0 2 0 0 0 0 0 0 0	8 8 64 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 1 0 0 1 1 1 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0
Number of routes: 3	Number of routes: 2
Route 1 x=7 y=0 d=0 e=0 x=6 y=0 d=1 e=1 x=5 y=0 d=2 e=2 x=4 y=0 d=3 e=3 x=3 y=0 d=4 e=4 x=2 y=0 d=5 e=5 x=1 y=0 d=6 e=6 x=0 y=0 d=7 e=7 x=0 y=1 d=8 e=8 x=0 y=2 d=9 e=9 x=0 y=3 d=10 e=10 x=0 y=4 d=11 e=11 x=0 y=5 d=12 e=12 x=0 y=6 d=13 e=13 x=0 y=7 d=14 e=14 x=1 y=1 d=7 e=23 x=1 y=2 d=8 e=24 x=1 y=3 d=9 e=25 x=1 y=4 d=10 e=26 x=1 y=5 d=11 e=27 x=1 y=6 d=12 e=28 x=1 y=7 d=13 e=29 x=2 y=1 d=6 e=38 x=2 y=2 d=7 e=39 x=2 y=3 d=8 e=40 x=2 y=4 d=9 e=41 x=2 y=5 d=10 e=42	Route 1 x=7 y=0 d=0 e=0 x=6 y=0 d=1 e=1 x=5 y=0 d=2 e=2 x=4 y=0 d=3 e=3 x=3 y=0 d=4 e=4 x=2 y=0 d=5 e=5 x=1 y=0 d=6 e=6 x=0 y=0 d=7 e=7 x=0 y=1 d=8 e=8 x=0 y=2 d=9 e=9 x=0 y=3 d=10 e=10 x=0 y=4 d=11 e=11 x=0 y=5 d=12 e=12 x=0 y=6 d=13 e=13 x=0 y=7 d=14 e=14 x=1 y=1 d=7 e=23 x=1 y=2 d=8 e=24 x=1 y=3 d=9 e=25 x=1 y=4 d=10 e=26 x=1 y=5 d=11 e=27 x=1 y=6 d=12 e=28 x=1 y=7 d=13 e=29 x=4 y=1 d=4 e=40 x=4 y=2 d=5 e=41 x=4 y=3 d=6 e=42 x=4 y=4 d=7 e=43 x=3 y=4 d=8 e=44

TestCase 1	TestCase 2
x=2 y=6 d=11 e=43	x=2 y=4 d=9 e=45
x=2 y=7 d=12 e=44	x=5 y=1 d=3 e=53
x=3 y=1 d=5 e=53	x=5 y=2 d=4 e=54
x=3 y=2 d=6 e=54	x=6 y=1 d=2 e=58
x=3 y=3 d=7 e=55	x=6 y=2 d=3 e=59
x=3 y=4 d=8 e=56	x=6 y=3 d=4 e=60
Route 2	Route 2
x=7 y=0 d=0 e=0	x=7 y=0 d=0 e=0
x=6 y=0 d=1 e=1	x=6 y=0 d=1 e=1
x=5 y=0 d=2 e=2	x=6 y=1 d=2 e=2
x=4 y=0 d=3 e=3	x=6 y=2 d=3 e=3
x=3 y=0 d=4 e=4	x=6 y=3 d=4 e=4
x=3 y=1 d=5 e=5	x=6 y=4 d=5 e=5
x=3 y=2 d=6 e=6	x=6 y=5 d=6 e=6
x=3 y=3 d=7 e=7	x=6 y=6 d=7 e=7
x=3 y=4 d=8 e=8	x=6 y=7 d=8 e=8
x=3 y=5 d=9 e=9	x=5 y=7 d=9 e=9
x=3 y=6 d=10 e=10	x=4 y=7 d=10 e=10
x=3 y=7 d=11 e=11	x=3 y=7 d=11 e=11
x=4 y=1 d=4 e=20	x=2 y=7 d=12 e=12
x=4 y=2 d=5 e=21	x=7 y=1 d=1 e=25
x=4 y=3 d=6 e=22	x=7 y=2 d=2 e=26
x=4 y=4 d=7 e=23	x=7 y=3 d=3 e=27
x=4 y=5 d=8 e=24	x=7 y=4 d=4 e=28
x=4 y=6 d=9 e=25	x=7 y=5 d=5 e=29
x=4 y=7 d=10 e=26	x=7 y=6 d=6 e=30
x=5 y=1 d=3 e=35	x=7 y=7 d=7 e=31
x=5 y=2 d=4 e=36	
x=5 y=3 d=5 e=37	
x=5 y=4 d=6 e=38	
x=5 y=5 d=7 e=39	
x=5 y=6 d=8 e=40	
x=5 y=7 d=9 e=41	
x=6 y=1 d=2 e=50	
x=6 y=2 d=3 e=51	
x=6 y=3 d=4 e=52	
x=6 y=4 d=5 e=53	
x=6 y=5 d=6 e=54	
x=6 y=6 d=7 e=55	
x=6 y=7 d=8 e=56	
Route 3	
x=7 y=0 d=0 e=0	
x=7 y=1 d=1 e=1	
x=7 y=2 d=2 e=2	
x=7 y=3 d=3 e=3	
x=7 y=4 d=4 e=4	
x=7 y=5 d=5 e=5	
x=7 y=6 d=6 e=6	
x=7 y=7 d=7 e=7	

Chương 3

KIỂM TRA VÀ ĐÁNH GIÁ

3.1 KIỂM TRA TÍNH ĐÚNG ĐẮN

3.1.1 PDFS OFFLINE

KIỂM TRA BẰNG THUẬT TOÁN

```
7 20
0 1 1
0 6 2
1 2 8
1 3 4
3 4 3
3 5 3
0 1 2 1 3 4 3 5 3 1 0 6 0
Route 1 (cost = 18): 0 1 2 1 0
Route 2 (cost = 16): 0 1 3 4 3 1 0
Route 3 (cost = 20): 0 1 3 5 3 1 0 6 0

F:\LamQuen\Bai1\x64\Debug\Bai1.exe (process 552) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Deb
le when debugging stops.
Press any key to close this window . . .
```

Hình 3.1: Kiểm tra bằng thuật toán cho TestCase 1 ($B = 20$)


```

7 18
0 1 1
0 6 2
1 2 8
1 3 4
3 4 3
3 5 3
0 1 2 1 3 4 3 5 3 1 0 6 0
Route 1 (cost = 18): 0 1 2 1 0
Route 2 (cost = 16): 0 1 3 4 3 1 0
Route 3 (cost = 16): 0 1 3 5 3 1 0
Route 4 (cost = 4): 0 6 0

C:\Users\hello\Desktop\PDFS\x64\Debug\Project1.exe (process 4252) exited with code 0 (0x0).
Press any key to close this window . . .

```

Hình 3.2: Kiểm tra bằng thuật toán cho TestCase 1 ($B = 18$)

Giá trị $B = 20$ (thỏa mãn theo yêu cầu). Có thể đặt $B=18$, nhưng nếu đặt như thế thì sẽ xuất hiện thêm 1 tuyến đường khác nữa. Cụ thể:

Nếu như ta đặt $B=18$ thì vẫn thỏa mãn, đúng tính chất nhưng nếu làm thế, thì thuật toán sẽ sinh thêm 1 tuyến đường mới là Route 4 (cost = 4): 0 6 0. Điều này bắt buộc ta phải về gốc để nạp một lần năng lượng nữa, nên sẽ làm tăng thêm chi phí, do đó việc đặt $B = 20$ là hoàn toàn hợp lý và khả thi.

```

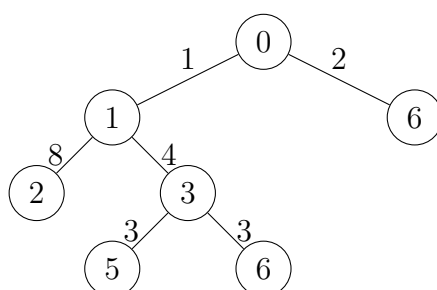
11 24
0 1 3
0 3 1
1 2 1
1 4 5
2 7 4
2 8 7
3 5 6
3 6 1
5 9 1
5 10 5
0 1 2 7 2 8 2 1 4 1 0 3 5 9 5 10 5 3 6 3 0
Route 1 (cost = 16): 0 1 2 7 2 1 0
Route 2 (cost = 22): 0 1 2 8 2 1 0
Route 3 (cost = 18): 0 1 4 1 0 3 0
Route 4 (cost = 16): 0 3 5 9 5 3 0
Route 5 (cost = 24): 0 3 5 10 5 3 0
Route 6 (cost = 4): 0 3 6 3 0

C:\Users\hello\Desktop\PDFS\x64\Debug\Project1.exe (process 8272) exited with code 0 (0x0).
Press any key to close this window . . .

```

Hình 3.3: Kiểm tra bằng thuật toán cho TestCase 2

KIỂM TRA THỦ CÔNG

Hình 3.4: Cây đồ thị ví dụ với $B = 20$

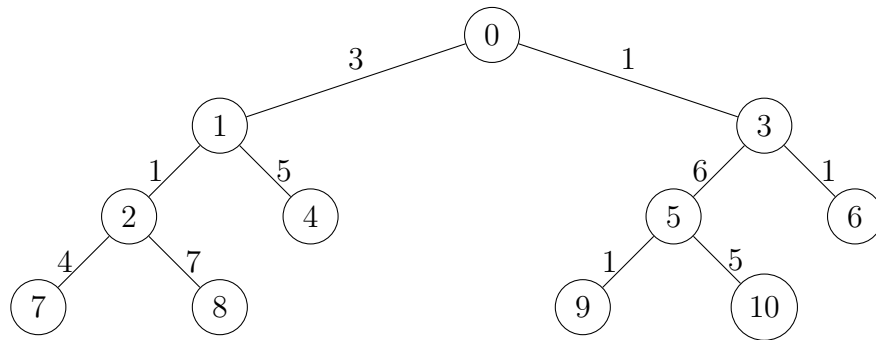
Theo như lý thuyết đã nói trên, ta xét hình 3.4 và có:

DFS: 0 1 2 1 3 4 3 5 3 1 0 6 0

Giá trị B thỏa mãn là: **B=20**

Dựa vào DFS và B, ta thực hiện chia các tuyến cho cây như sau:

- Lượt 1: 0 1 2 1 0, năng lượng tiêu tốn là 18
- Lượt 2: 0 1 3 4 3 1 0, năng lượng tiêu tốn là 16
- Lượt 3: 0 1 3 5 3 1 0 6 0, năng lượng tiêu tốn là 20



Hình 3.5: Cây đồ thị với B=24

Theo như lý thuyết đã đề cập phía trên, ta xét hình 3.5 và có:

DFS: 0 1 2 7 2 8 2 1 4 1 0 3 5 9 5 10 5 3 6 3 0

Giá trị B thỏa mãn là: **B=24**

Dựa vào DFS và B, ta thực hiện chia các tuyến cho cây như sau:

- Lượt 1: 0 1 2 7 2 1 0, năng lượng tiêu tốn là 16
- Lượt 2: 0 1 2 8 2 1 0, năng lượng tiêu tốn là 22
- Lượt 3: 0 1 4 1 0 3 0, năng lượng tiêu tốn là 18
- Lượt 4: 0 3 5 9 5 3 0, năng lượng tiêu tốn là 16
- Lượt 5: 0 3 5 10 5 3 0, năng lượng tiêu tốn là 24
- Lượt 6: 0 3 6 3 0, năng lượng tiêu tốn là 4

NHẬN XÉT

Kết quả kiểm tra thuật toán hoàn toàn trùng khớp với kết quả kiểm tra thủ công.

Cụ thể, các giá trị DFS và B (B = 20 cho TestCase 1, B = 24 cho TestCase 2) đều được tính toán chính xác.

Kiểm tra thủ công giúp ta làm rõ tính logic của thuật toán thông qua việc chia nhỏ từng tuyến và tính toán năng lượng tiêu hao (ví dụ: Lượt 1, 2, 3 ở TestCase 1). Điều này giúp ta hiểu sâu hơn về cách thuật toán chia tuyến và chọn đường đi.

⇒ **PDFS offline đã thể hiện được tính đúng đắn của thuật toán**

Bên cạnh đó ta thấy: PDFS Offline phù hợp với các hệ thống yêu cầu dữ liệu đầu vào không thay đổi, cho phép lập kế hoạch tuyến đường tĩnh và tiết kiệm thời gian triển khai. Hơn thế việc kiểm tra thủ công tốn nhiều công sức và dễ sai sót nếu cấu trúc cây/đồ thị phức tạp.

3.1.2 PDFS ONLINE

KIỂM TRA BẰNG THUẬT TOÁN

TestCase 1	TestCase 2
8 8 64 0 2 0 0 0 0 0 0 0	8 8 64 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 1 0 0 1 1 1 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0
Number of routes: 3 Route 1 x=7 y=0 d=0 e=0 x=6 y=0 d=1 e=1 x=5 y=0 d=2 e=2 x=4 y=0 d=3 e=3 x=3 y=0 d=4 e=4 x=2 y=0 d=5 e=5 x=1 y=0 d=6 e=6 x=0 y=0 d=7 e=7 x=0 y=1 d=8 e=8 x=0 y=2 d=9 e=9 x=0 y=3 d=10 e=10 x=0 y=4 d=11 e=11 x=0 y=5 d=12 e=12 x=0 y=6 d=13 e=13 x=0 y=7 d=14 e=14 x=1 y=1 d=7 e=23 x=1 y=2 d=8 e=24 x=1 y=3 d=9 e=25 x=1 y=4 d=10 e=26 x=1 y=5 d=11 e=27 x=1 y=6 d=12 e=28 x=1 y=7 d=13 e=29 x=2 y=1 d=6 e=38 x=2 y=2 d=7 e=39 x=2 y=3 d=8 e=40 x=2 y=4 d=9 e=41 x=2 y=5 d=10 e=42 x=2 y=6 d=11 e=43 x=2 y=7 d=12 e=44 x=3 y=1 d=5 e=53 x=3 y=2 d=6 e=54 x=3 y=3 d=7 e=55 x=3 y=4 d=8 e=56 Route 2 x=7 y=0 d=0 e=0	Number of routes: 2 Route 1 x=7 y=0 d=0 e=0 x=6 y=0 d=1 e=1 x=5 y=0 d=2 e=2 x=4 y=0 d=3 e=3 x=3 y=0 d=4 e=4 x=2 y=0 d=5 e=5 x=1 y=0 d=6 e=6 x=0 y=0 d=7 e=7 x=0 y=1 d=8 e=8 x=0 y=2 d=9 e=9 x=0 y=3 d=10 e=10 x=0 y=4 d=11 e=11 x=0 y=5 d=12 e=12 x=0 y=6 d=13 e=13 x=0 y=7 d=14 e=14 x=1 y=1 d=7 e=23 x=1 y=2 d=8 e=24 x=1 y=3 d=9 e=25 x=1 y=4 d=10 e=26 x=1 y=5 d=11 e=27 x=1 y=6 d=12 e=28 x=1 y=7 d=13 e=29 x=4 y=1 d=4 e=40 x=4 y=2 d=5 e=41 x=4 y=3 d=6 e=42 x=4 y=4 d=7 e=43 x=3 y=4 d=8 e=44 x=2 y=4 d=9 e=45 x=5 y=1 d=3 e=53 x=5 y=2 d=4 e=54 x=6 y=1 d=2 e=58 x=6 y=2 d=3 e=59 x=6 y=3 d=4 e=60 Route 2 x=7 y=0 d=0 e=0

TestCase 1	TestCase 2
x=6 y=0 d=1 e=1	x=6 y=0 d=1 e=1
x=5 y=0 d=2 e=2	x=6 y=1 d=2 e=2
x=4 y=0 d=3 e=3	x=6 y=2 d=3 e=3
x=3 y=0 d=4 e=4	x=6 y=3 d=4 e=4
x=3 y=1 d=5 e=5	x=6 y=4 d=5 e=5
x=3 y=2 d=6 e=6	x=6 y=5 d=6 e=6
x=3 y=3 d=7 e=7	x=6 y=6 d=7 e=7
x=3 y=4 d=8 e=8	x=6 y=7 d=8 e=8
x=3 y=5 d=9 e=9	x=5 y=7 d=9 e=9
x=3 y=6 d=10 e=10	x=4 y=7 d=10 e=10
x=3 y=7 d=11 e=11	x=3 y=7 d=11 e=11
x=4 y=1 d=4 e=20	x=2 y=7 d=12 e=12
x=4 y=2 d=5 e=21	x=7 y=1 d=1 e=25
x=4 y=3 d=6 e=22	x=7 y=2 d=2 e=26
x=4 y=4 d=7 e=23	x=7 y=3 d=3 e=27
x=4 y=5 d=8 e=24	x=7 y=4 d=4 e=28
x=4 y=6 d=9 e=25	x=7 y=5 d=5 e=29
x=4 y=7 d=10 e=26	x=7 y=6 d=6 e=30
x=5 y=1 d=3 e=35	x=7 y=7 d=7 e=31
x=5 y=2 d=4 e=36	
x=5 y=3 d=5 e=37	
x=5 y=4 d=6 e=38	
x=5 y=5 d=7 e=39	
x=5 y=6 d=8 e=40	
x=5 y=7 d=9 e=41	
x=6 y=1 d=2 e=50	
x=6 y=2 d=3 e=51	
x=6 y=3 d=4 e=52	
x=6 y=4 d=5 e=53	
x=6 y=5 d=6 e=54	
x=6 y=6 d=7 e=55	
x=6 y=7 d=8 e=56	
Route 3	
x=7 y=0 d=0 e=0	
x=7 y=1 d=1 e=1	
x=7 y=2 d=2 e=2	
x=7 y=3 d=3 e=3	
x=7 y=4 d=4 e=4	
x=7 y=5 d=5 e=5	
x=7 y=6 d=6 e=6	
x=7 y=7 d=7 e=7	

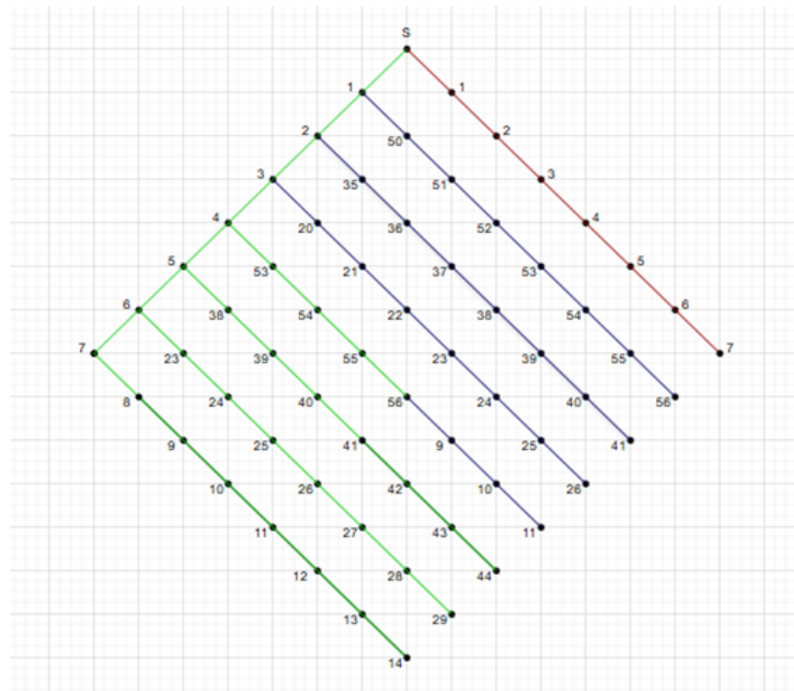
KIỂM TRA THỦ CÔNG

7	8	9	10	11	12	13	14
6	23	24	25	26	27	28	29
5	38	39	40	41	42	43	44
4	53	54	55	56	9	10	11
3	20	21	22	23	24	25	26
2	35	36	37	38	39	40	41
1	50	51	52	53	54	55	56
2	1	2	3	4	5	6	7

Hình 3.6: Bảng năng lượng của TestCase 1

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)	(3,7)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)
(5,0)	(5,1)	(5,2)	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)
(6,0)	(6,1)	(6,2)	(6,3)	(6,4)	(6,5)	(6,6)	(6,7)
(7,0)	(7,1)	(7,2)	(7,3)	(7,4)	(7,5)	(7,6)	(7,7)

Hình 3.7: Bảng tọa độ của TestCase 1



Hình 3.8: Mô phỏng TestCase 1

Giá trị B thỏa mãn là: $B = 64$

Dựa vào bảng năng lượng từ Hình 3.6 ta có thể thấy robot có 3 lần quay về điểm xuất phát để nạp năng lượng, nên được chia thành 3 Route như sau:

Route 1: (7,0) (6,0) ... (0,0) (0,1) ... (0,7) (1,1) (1,2) ... (1,7) (2,1) (2,2) ... (2,7) (3,1) ... (3,4)

Route 2: (7,0) (6,0) ... (3,0) (3,1) ... (3,7) (4,1) (4,2) ... (4,7) (5,1) (5,2) ... (5,7) (6,1) (6,2) ... (6,7)

Route 3: (7,0) (7,1) (7,2) (7,3) (7,4) (7,5) (7,6) (7,7)

Chú thích: Năng lượng e dựa vào đường đi giữa các Route và bảng năng lượng (Hình 3.6).

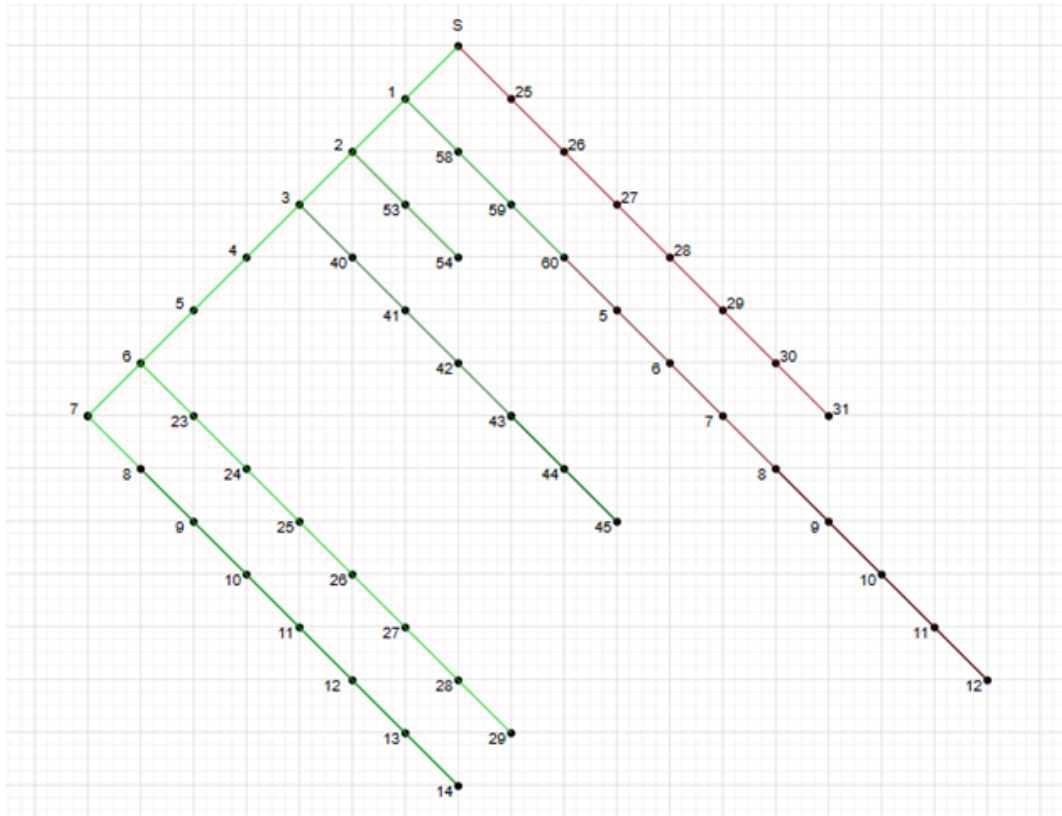
Khoảng cách giữa điểm hiện tại và điểm bắt đầu ta có thể thấy rõ ở Hình 3.8

7	8	9	10	11	12	13	14
6	23	24	25	26	27	28	29
5	X	X	X	45	X	X	12
4	X	X	X	44	X	X	11
3	40	41	42	43	X	X	10
2	53	54	X	X	X	X	9
1	58	59	60	5	6	7	8
2	25	26	27	28	29	30	31

Hình 3.9: Bảng năng lượng của TestCase 2

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)
(2,0)	X	X	X	(2,4)	X	X	(2,7)
(3,0)	X	X	X	(3,4)	X	X	(3,7)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)	X	X	(4,7)
(5,0)	(5,1)	(5,2)	X	X	X	X	(5,7)
(6,0)	(6,1)	(6,2)	(6,3)	(6,4)	(6,5)	(6,6)	(6,7)
(7,0)	(7,1)	(7,2)	(7,3)	(7,4)	(7,5)	(7,6)	(7,7)

Hình 3.10: Bảng tọa độ TestCase 2



Hình 3.11: Mô phỏng TestCase 2

Giá trị B thỏa mãn là: $B = 64$

Dựa vào bảng năng lượng từ Hình 3.9 ta có thể thấy robot có 2 lần quay về điểm xuất phát để nạp năng lượng, nên được chia thành 2 Route như sau:

Route 1: (7,0) (6,0) ... (0,0) (0,1) ... (0,7) (1,1) (1,2) ... (1,7) (4,1) (4,2) (4,3) (4,4) (3,4) (2,4) (5,1) (5,2) (6,1) (6,2) (6,3)

Route 2: (7,0) (6,0) (6,1) ... (6,7) (5,7) (4,7) (3,7) (2,7) (7,1) (7,2) (7,3) (7,4) (7,5) (7,6) (7,7)

Chú thích: Năng lượng e dựa vào đường đi giữa các Route và bảng năng lượng (Hình 3.9).

Khoảng cách d giữa điểm hiện tại và điểm bắt đầu ta có thể thấy rõ ở Hình 3.11

NHẬN XÉT

Kết quả từ thuật toán PDFS Online ở cả hai TestCase đều khớp với kiểm tr thủ công. Số lượng tuyến đường, giá trị năng lượng $B = 64$ và cách phân chia các route đều được xác định chính xác, thể hiện qua việc robot quay về điểm gốc đúng số lần dự kiến (3 lần cho TestCase 1 và 2 lần cho TestCase 2).

Thuật toán tối ưu hóa được việc sử dụng năng lượng B bằng cách phân chia các tuyến đường hợp lý, giảm thiểu số lần quay về gốc không cần thiết. Ví dụ, ở TestCase 1, việc chọn $B = 64$ giúp robot hoàn thành nhiệm vụ với 3 route mà không cần tăng thêm chi phí năng lượng.

Thuật toán thể hiện rõ cấu trúc lưới và đường đi của robot thông qua các bảng tọa độ và năng lượng (Hình 3.6, 3.7, 3.9, 3.10). Điều này cho phép đánh giá trực quan hiệu quả của thuật toán trước khi triển khai thực tế. Ngoài ra việc xử lý các ma trận phức tạp hơn (ví dụ: kích thước lớn hoặc có nhiều chướng ngại vật) cần được nghiên cứu thêm.

3.2 ĐÁNH GIÁ THUẬT TOÁN

3.2.1 Ưu điểm của PDFS Offline và PDFS Online

Tiêu chí	PDFS Offline
Độ chính xác và độ tin cậy đối với thuật toán	Độ chính xác cao vì kết quả thuật toán khớp với kiểm tra thủ công
Mô phỏng và kiểm tra trước	Có thể mô phỏng cấu trúc cây để đánh giá trước khi thực hiện thuật toán
Phù hợp với dữ liệu cố định	Thuật toán thực hiện hiệu quả khi dữ liệu đầu vào(cây) không thay đổi, không cập nhật liên tục

Bảng 3.2: Ưu điểm của PDFS Offline

Tiêu chí	PDFS Online
Xử lý dữ liệu động	Phù hợp với các bài toán có thay đổi trong thời gian thực và có cấu trúc thay đổi liên tục
Thời gian phản hồi nhanh	Không yêu cầu biết toàn bộ dữ liệu trước khi xử lý, giúp giảm độ trễ và tăng tốc độ phản hồi
Linh hoạt trong môi trường không ổn định	Giải quyết hiệu quả các bài toán có điều kiện ràng buộc thay đổi. Tự động điều chỉnh lộ trình dựa trên thông tin mới nhất, đảm bảo luôn là tối ưu
Tiết kiệm tài nguyên	Xử lý từng phần dữ liệu khi cần thiết, giảm thiểu việc lưu trữ và tính toán dư thừa. Phù hợp với robot có hạn chế về bộ nhớ và năng lượng
Khả năng mở rộng	Dễ dàng tích hợp vào các hệ thống phân tán hoặc song song

Bảng 3.3: Ưu điểm của PDFS Online

3.2.2 Nhược điểm của PDFS Offline và PDFS Online

Tiêu chí	PDFS Offline
Thiếu linh hoạt với dữ liệu động	Yêu cầu đầu vào cố định và không thể tự điều chỉnh khi có thay đổi. Mọi thay đổi đều cần phải chạy lại thuật toán từ đầu, làm tốn thời gian và tài nguyên
Hiệu suất giảm với dữ liệu lớn	Khi kích thước tăng, thời gian tính toán và bộ nhớ cần thiết để duyệt cây và phân chia tuyến đường tăng đáng kể, đặc biệt đối với cấu trúc cây phức tạp hoặc khoảng cách xa từ gốc đến lá
Phụ thuộc vào giá trị B	Chọn giá trị B tối ưu đòi hỏi kiểm tra thủ công hoặc mô phỏng trước. Nếu B không được chọn chính xác, thuật toán có thể làm tăng số tuyến, làm tăng chi phí hoặc không đáp ứng yêu cầu năng lượng
Tốn công sức kiểm tra thủ công	Để đảm bảo độ chính xác, cần mô phỏng và kiểm tra thủ công từng trường hợp. Quá trình này tốn thời gian, đặc biệt khi số lượng TestCase lớn hoặc cấu trúc phức tạp
Không phù hợp với thời gian thực	Chỉ xử lý dữ liệu tĩnh và không có khả năng phản hồi tức thời. Nên làm hạn chế ứng dụng trong các hệ thống yêu cầu xử lý nhanh

Bảng 3.4: Nhược điểm của PDFS Offline

Tiêu chí	PDFS Online
Phụ thuộc vào tốc độ cập nhật dữ liệu	Khi dữ liệu đầu vào thay đổi quá nhanh, thuật toán có thể không theo kịp, dẫn đến kết quả lỗi thời gian hoặc không chính xác
Thiếu cái nhìn tổng quan	Xử lý từng phần dữ liệu, thuật toán dễ bỏ qua các mối liên kết quan trọng trong toàn bộ đồ thị. Điều này có thể dẫn đến quyết định cục bộ thay vì tối ưu toàn cục
Độ phức tạp quản lý trạng thái	Duy trì trạng thái liên tục giữa các phiên xử lý đòi hỏi quản lý bộ nhớ phức tạp hơn, đặc biệt khi dữ liệu không ổn định. Điều này làm tăng nguy cơ rò rỉ bộ nhớ hoặc lỗi đồng bộ hóa
Hiệu suất giảm trên quy mô lớn	Khi áp dụng đồ thị phức tạp, có thể gặp khó khăn trong việc đồng bộ hóa, dẫn đến độ trễ cao hoặc tiêu tốn tài nguyên
Nhạy cảm với chất lượng dữ liệu đầu vào	Nếu dữ liệu thời gian thực không đáng tin cậy, thuật toán dễ đưa ra kết quả sai lầm
Khó debug và kiểm thử	Do tính chất động và không dự đoán trước nên việc theo dõi lỗi hoặc kiểm tra thủ công trở nên khó khăn, đòi hỏi phải có công cụ để giám sát quá trình xử lý

Bảng 3.5: Nhược điểm của PDFS Online

3.3 KẾT LUẬN

Cả hai phương pháp đều cho kết quả chính xác, nhưng PDFS Online phù hợp hơn cho bài toán duyệt trên đồ thị lưới, trong khi PDFS Offline tối ưu cho cấu trúc cây.

PDFS Offline tập trung vào việc chọn giá trị B phù hợp để giảm chi phí quay về gốc (ví dụ: $B = 20$ thay vì $B = 18$ ở TestCase 1), trong khi PDFS Online tối ưu dựa trên ma trận năng lượng cố định.

PDFS Online phù hợp cho robot di chuyển trong môi trường có sẵn thông tin lưới, còn PDFS Offline thích hợp cho hệ thống cố định (ví dụ: mạng cảm biến).

- Vậy cả hai phương pháp đều thể hiện ưu và nhược điểm riêng, nhưng cần kết hợp linh hoạt tùy vào trường hợp cụ thể. Việc kiểm giữa thuật toán và thử công đã khẳng định tính đúng đắn của cả 2 phương pháp, đồng thời làm rõ bản chất của thuật toán PDFS trong các tình huống khác nhau.

Chương 4

Mở rộng cho PDFS OFFLINE

4.1 Giới thiệu

Trong quá trình kiểm thử và thực hiện trên đồ thị cây của thuật toán PDFS offline, ta thấy rằng có những trường hợp của kết quả thuật toán mà năng lượng tiêu hao cho mỗi tuyến không đạt tới B thường khá lớn. Xem xét lại hai testcase trên [mục 2.1.2](#), ta thấy

Testcase 1: Có duy nhất 1 tuyến 3 là đạt tới giá trị B, 2 tuyến 1 và 2 có năng lượng dư lần lượt là: 2 và 4

Testcase 2: Có duy nhất 1 tuyến 5 là đạt tới giá trị B, 5 tuyến còn lại là dư thừa năng lượng.

Theo như tài liệu [1], thì không đề cập về việc dùng năng lượng dư thừa để tối ưu số tuyến mà chỉ nói đến vấn đề là đảm bảo đúng tính chất công thức 1.1, do đó lượng dư thừa sau mỗi tuyến bị bỏ phí. Giả sử nếu như ứng dụng trong thực tế khám phá thì chi phí cho lượng dư thừa này khá lớn, từ đó dẫn đến việc tổn hao về tài chính khám phá. Nhận thấy vấn đề trên nên nhóm đã quyết định cải tiến thuật toán để tối ưu hơn về lượng dư thừa này.

4.2 Thiết lập và giải quyết vấn đề

4.2.1 Cơ sở lý thuyết

- **Nền tảng 1:** Nói về các lý thuyết về *giá trị của B, môi trường khám phá, bài toán ECTC, các kiến thức liên quan* đã được nói đến trong tài liệu [1]
- **Nền tảng 2:** Các trường hợp mà thuật toán sắp cải tối bị giới hạn:
 1. Không có dư thừa trong quá trình khám phá
Cụ thể: Giá trị B chia hết cho trọng số của các cạnh
 2. Lượng dư thừa quá bé so với trọng số của cây
 3. Lượng dư thừa đã được dùng ở một tuyến nào đó, và khi tiếp một tuyến khác thì lại phải quay về làm cho không tối ưu được số tuyến như mong muốn. Tuy nhiên khi quay về và khám phá thì một điều có thể khẳng định tuyến đó là tuyến cuối cùng trong quá trình khám phá.
- **Nền tảng 3:** Ở cải tiến này ta có thêm 1 giá trị C. C sẽ lưu trữ dư thừa ở tuyến, và sẽ cộng dồn vào. Công thức của C:

$$C = \sum_{i=1}^k B - c_i$$

Giải thích công thức:

B: giá trị giới hạn

c_i : chi phí cho tuyến i

k: số tuyến hiện tại đang duyệt (max: $n-1$)

- **Nền tảng 4:** Trong [1], đã nói đến *giá trị B là giới hạn năng lượng cho quá trình khám phá, có thể là pin*. Vậy thì giá trị C, ta dùng như thế nào khi B còn dư lại.

Ý tưởng: Robot ngoài pin mang giá trị B, thì còn một viên pin rỗng để tích trữ năng lượng. Viên pin này có dung tích đủ lớn để lưu lượng dư thừa

Điều kiện: $C \geq B$, nghĩa là robot sẽ mang theo 1 viên pin rỗng có sức chứa ngang viên pin hiện tại mà robot đang dùng. Qua các tuyến thì lượng dư thừa sẽ được nạp vào C.

Lưu ý:

→ Giá trị dư thừa sẽ giúp ta *giảm h tuyến, tiết kiệm được h lần quay về*. *Giảm được h lần chi phí*. Đây là điều mà hầu hết các nhà khám phá đều muốn đạt được

4.2.2 Quá trình thực hiện

1. Bắt đầu tại gốc r
2. Thực hiện chia tuyến như trong [1], nhưng có 1 lưu ý là duyệt DFS sẽ khác với DFS khi chưa cải tiến, xem thêm tại mục [kiểm tra thủ công](#)
3. Sau tuyến đầu tiên, giá trị C đã được gán vào với $C = B - c_1$. Lúc này robot đang ở gốc
4. Xem xét là có nhánh nào ở gốc có thể duyệt được thì thêm vào tuyến đầu tiên này, nếu không tiếp tục 1 tuyến mới
5. Ở tuyến i , ta vẫn duyệt tương tự theo như [1], nhưng khi không thể đi tiếp vì nếu đi tiếp sẽ vượt qua năng lượng B ban đầu, thì bây giờ ta dùng giá trị C. Nếu C có thể duyệt thêm một nhánh nào thì ta thêm nhánh đó vào tuyến này
6. Lặp lại bước 5 cho đến khi duyệt hết cây

4.3 Thuật toán

Trong thuật toán, ta đặt C là **surPlus**.

Về cấu trúc: tương tự mã giả của PDFS offline. Nhưng có thêm 2 hàm mới

Hàm `exploreWithSurplus(Node, double)`

Hàm này sẽ kiểm tra xem: các nút con của nút hiện tại đang dùng, có nút nào có thể khám phá được không, nếu được thì thêm vào tuyến

```

1 for all (v, w) in u.children do
2   if not v.visited && weight*2<=surPlus then
3     Add child.id to currentRoute, fullDFS;
4     routeEnergy+=weight*2;
5     exploreWithSurplus(child, surPlus-(weight*2));
6     Add currentNode.id to currentRoute, fullDFS;
7   endif
8 end for

```

Hàm DFS

```

1 Function DFS(u)
2   for all (v, w) in u.children do
3     if not v.visited then
4       if routeEnergy + w + (currentLength + w) > B + carriedSurplus then
5         surplus <- B + carriedSurplus - (routeEnergy + currentLength)
6         if surplus > 0 then
7           exploreWithSurplus(u, surplus)
8         end if
9         distToRoot <- currentLength
10        routeEnergy <- routeEnergy + distToRoot
11        actualSurplus <- B + carriedSurplus - routeEnergy
12        Add actualSurplus to surplusEnergy
13        carriedSurplus <- actualSurplus
14        for i <- path.size - 2 downto 0 do
15          Add path[i].first.id to currentRoute
16        end for
17        Add currentRoute to routes
18        Add routeEnergy to routeCosts
19        routeCount <- routeCount + 1
20        currentRoute <- empty
21        Add root.id to currentRoute
22        routeEnergy <- 0
23        newDist <- 0
24        for i <- 1 to path.size - 1 do
25          parentNode <- path[i - 1].first
26          curNode <- path[i].first
27          wt <- path[i].second
28          newDist <- newDist + wt
29          routeEnergy <- routeEnergy + wt
30          sumEnergy <- sumEnergy + newDist
31          Add curNode.id to currentRoute
32        end for
33        currentLength <- newDist
34      end if
35      v.visited <- true
36      Add v.id to fullDFS
37      currentLength <- currentLength + w
38      routeEnergy <- routeEnergy + w
39      sumEnergy <- sumEnergy + currentLength
40      Add v.id to currentRoute
41      Add (v, w) to path
42      DFS(v)
43      Add u.id to fullDFS
44      Remove last element from path
45      currentLength <- currentLength - w
46      routeEnergy <- routeEnergy + w
47      Add u.id to currentRoute
48    end if
49  end for
50 End Function
51
52 // Initial call
53 root.visited <- true
54 DFS(root)
55
56 // Finalize last route if any
57 if currentRoute.size > 1 then
58   distToRoot <- currentLength
59   routeEnergy <- routeEnergy + distToRoot
60   finalSurplus <- B + carriedSurplus - routeEnergy
61   if finalSurplus > 0 then
62     exploreWithSurplus(root, finalSurplus)

```

```

63     end if
64     Add finalSurplus to surplusEnergy
65     carriedSurplus <- finalSurplus
66     for i <- path.size - 2 downto 0 do
67         Add path[i].first.id to currentRoute
68     end for
69     Add currentRoute to routes
70     Add routeEnergy to routeCosts
71 end if
72
73 // Output full DFS traversal
74 for each id in fullDFS do
75     print id
76 end for
77
78 // Output route info
79 for i <- 0 to routes.size - 1 do
80     print "Route", i + 1, "(cost =", routeCosts[i], ", surplus =",
      surplusEnergy[i], "):"
81     for j <- 0 to routes[i].size - 1 do
82         print routes[i][j]
83     end for
84 end for

```

4.4 Triển khai code và đánh giá

4.4.1 Triển khai

Code đã đính kèm theo link: [Code](#)

4.4.2 Minh họa testcase và kiểm tra thủ công

Chạy thuật toán

Input:

Dòng 1: n B (số nút, năng lượng).

Dòng 2-n: cạnh có trọng số.

Output:

Dòng 1: DFS quay lui.

Tiếp theo: các tuyến.

TestCase 1	TestCase 2
------------	------------

10 26	11 24
0 1 2	0 1 3
0 2 3	0 3 1
1 3 4	1 2 1
1 4 1	1 4 5
2 5 5	2 7 4
2 6 1	2 8 7
5 7 2	3 5 6
5 8 2	3 6 1
8 9 3	5 9 1
	5 10 5
0 1 3 1 4 1 0 2 6 2 5 7 5 8 9 8 5 2 0	0 1 2 7 2 8 2 1 4 1 0 3 5 9 5 10 5 3 6 3 0
Route 1 (cost = 22, surplus = 4): 0 1 3 1 4 1 0 2 6 2 0	Route 1 (cost = 16, surplus = 8): 0 1 2 7 2 1 0
Route 2 (cost = 30, surplus = 0): 0 2 5 7 5 8 9 8 5 2 0	Route 2 (cost = 32, surplus = 0): 0 1 2 8 2 1 4 1 0
	Route 3 (cost = 16, surplus = 8): 0 3 5 9 5 3 0
	Route 4 (cost = 26, surplus = 6): 0 3 5 10 5 3 6 3 0

```

10 26
0 1 2
0 2 3
1 3 4
1 4 1
2 5 5
2 6 1
5 7 2
5 8 2
8 9 3
0 1 3 1 4 1 0 2 6 2 5 7 5 8 9 8 5 2 0
Route 1 (cost = 22, surplus = 4): 0 1 3 1 4 1 0 2 6 2 0
Route 2 (cost = 30, surplus = 0): 0 2 5 7 5 8 9 8 5 2 0

```

Hình 4.1: TestCase 1 khi đã cải tiến

```

10 26
0 1 2
0 2 3
1 3 4
1 4 1
2 5 5
2 6 1
5 7 2
5 8 2
8 9 3
0 1 3 1 4 1 0 2 5 7 5 8 9 8 5 2 6 2 0
Route 1 (cost = 20): 0 1 3 1 4 1 0 2 0
Route 2 (cost = 24): 0 2 5 7 5 8 5 2 0
Route 3 (cost = 26): 0 2 5 8 9 8 5 2 0
Route 4 (cost = 8): 0 2 6 2 0

```

Hình 4.2: TestCase 1 khi chưa cải tiến

```

11 24
0 1 3
0 3 1
1 2 1
1 4 5
2 7 4
2 8 7
3 5 6
3 6 1
5 9 1
5 10 5
0 1 2 7 2 8 2 1 4 1 0 3 5 9 5 10 5 3 6 3 0
Route 1 (cost = 16, surplus = 8): 0 1 2 7 2 1 0
Route 2 (cost = 32, surplus = 0): 0 1 2 8 2 1 4 1 0
Route 3 (cost = 16, surplus = 8): 0 3 5 9 5 3 0
Route 4 (cost = 26, surplus = 6): 0 3 5 10 5 3 6 3 0

```

Hình 4.3: TestCase 2 khi đã cải tiến

Kiểm tra thủ công

Như đã nói lúc ban đầu, tuyến DFS cải tiến này sẽ khác với tuyến DFS trước khi cải tiến ở việc

Testcase 1: Cây mô phỏng như sau:

Tuyến DFS khi chưa cải tiến Hình 4.1	Tuyến DFS khi đã cải tiến Hình 4.2
0 1 3 1 4 1 0 2 5 7 5 8 9 8 5 2 6 2 0	0 1 3 1 4 1 0 2 6 2 5 7 5 8 9 8 5 2 0

Sự khác biệt: Khi chưa cải tiến là 0 2 5 7 ... 2 6 2 0, còn khi đã cải tiến thì tuyến DFS lúc này là 0 2 6 2 5 7 5 ... Lý do có sự thay đổi này, chính là bởi vì trong quá trình duyệt, lượng dư thừa có thể duyệt thêm cạnh 2-6, đây là điều mà DFS khi chưa cải tiến không thể thực hiện được. Chính nhờ việc duyệt thêm cạnh 2-6 này mà đã giúp ta giảm số tuyến từ 4 tuyến xuống còn 2 tuyến, giúp ta giảm tới 2 lần chi phí, 2 lần quay về giúp tiết kiệm rất nhiều thời gian công sức.

Việc thực hiện chia tuyến như sau:

Tuyến 1: Route 1 (cost = 22, surplus = 4): 0 1 3 1 4 1 0 2 6 2 0

Ta có thể thấy ngay khi mà duyệt thêm được cạnh 2-6 thì tuyến đầu tiên của ta đã thay đổi, ban đầu nếu chưa được cải tiến thì chi phí là 20, xem hình... còn bây giờ là 22, nhưng vẫn đảm bảo đúng tính chất của PDFS offline

Tuyến 2: Route 2 (cost = 30, surplus = 0): 0 2 5 7 5 8 9 8 5 2 0

Ta nhận thấy ở tuyến 2 này, có sự thay đổi rõ rệt về cả chi phí lẫn lượng dư, nếu như theo cách cũ thì ta chỉ có thể đạt đến 26 nhưng mà khi cải tiến giá trị vượt qua 26 vẫn không ảnh hưởng gì. Cụ thể hơn ở tuyến đầu ta dư được 4, hơn thế khi duyệt tuyến thứ 2 thì lượng dư này lại đủ để duyệt thêm cho nhánh cuối cùng, do đó mà số tuyến ta giảm được tới 50%. Giả sử như việc tuyến DFS khi chưa cải tiến giống như DFS cải tiến thì vẫn chỉ có thể giảm được 1 tuyến vì robot nhất định phải quay về để nạp năng lượng khám phá thêm tuyến cuối, nói đúng ra là cạnh cuối cùng, nhưng nhờ vào cải tiến mà ta không cần quay về vẫn có thể khám phá được toàn bộ cây.

TestCase 2: Tương tự testcase 1, nhờ vào tối ưu lượng dư mà dẫn đến việc duyệt DFS sẽ khác, mà khi duyệt DFS khác thì năng lượng lẫn chi phí đều được giảm đáng kể, xem 2 hình 3.3, 4.3. Cụ thể theo testcase 2 thì ta được giảm 2 tuyến so với PDFS truyền thống. Kèm theo link còn có các testcase khác, chúng ta có thể tự kiểm tra và nhận xét kỹ hơn nữa. Link testcase: [TestCase](#)

4.4.3 Đánh giá và ứng dụng tương lai

Thông qua mục [4.4.2](#), ta có thể thấy rằng thuật toán đã thể hiện được tính đúng đắn và đáp ứng được yêu cầu, mục đích ban đầu: tối ưu năng lượng, tiết kiệm chi phí. Theo như 2 testcase trên thì ta đã tiết kiệm được tới 50% chi phí cho việc khám phá kể cả thời gian lẫn công sức. Và hi vọng ý tưởng này sẽ được ứng dụng trong thực tế để tiết kiệm cho quá trình khám phá và thám hiểm trong tương lai.

DANH MỤC TÀI LIỆU THAM KHẢO

Tiếng Anh:

- [1] Shantanu Das, Przemysław Uznanski, Dariusz Dereniowski *Energy Constrained Depth First Search.*
- [2] Gokarna Sharma Ayan Dutta. *A Constant-Factor Approximation Algorithm for Online Coverage Path Planning with Energy Constrain*

Website:

- [1] [GitHub - PDFS](#)
- [2] [Viblo Asia](#)