به نام خدا

پروژه اول درس آزمایشگاه هوش مصنوعی

استاد راهنما : استاد زاد ضیابری

گردآورنده : مهدی شیرازی

بهار **1404**

چکیده

این گزارش به طراحی و پیاده‌سازی یک سیستم هوشمند مسیریابی برای کاوشگر مریخی "آستروبات" به عنوان پروژه اول می‌پردازد. سیستم توسعه داده شده با استفاده از الگوریتم جستجوی اول سطح (BFS) قادر به یافتن بهینه‌ترین مسیر حرکت کاوشگر در یک محیط ناشناخته مریخ با در نظر گرفتن موانع طبیعی است.

در طراحی و پیاده سازی این پروژه از زبان برنامه نویسی پایتون نسخه 3.12.6 استفاده شده است و همچنین از کتابخانه کاربردی collections برای پیاده سازی صف مورد نیاز الگوریتم BFS استفاده شده است.

**در ادامه به بررسی کامل پروژه می پردازیم ...**

**پروژه اول**

**مسئله اصلی این پروژه شامل :**

- یک شبکه ۵×۵ سلولی که محیط فرود کاوشگر را شبیه‌سازی می‌کند

- سلول‌های با مقدار ۰ نشان‌دهنده مناطق قابل عبور

- سلول‌های با مقدار ۱ نشان‌دهنده موانع غیرقابل عبور (صخره‌ها)

- نقطه شروع: سلول (0 و 0)

- نقطه هدف: سلول (4 و 4)

**- الگوریتم BFS به دلایل زیر برای این مسئله انتخاب شده است:**

۱. تضمین یافتن کوتاه‌ترین مسیر در صورت وجود مسیر ممکن

۲. پیاده‌سازی ساده و کارآمد با استفاده از ساختار داده صف

۳. عدم نیاز به اطلاعات قبلی از محیط

۴. مناسب برای شبکه‌های با ابعاد محدود

**نتایج نشان می‌دهد که سیستم توسعه داده شده قادر است:**

- در زمان مناسب (کمتر از ۱ ثانیه) مسیر بهینه را پیدا کند

- تمام محدودیت‌های حرکتی را رعایت نماید

- در شرایط مختلف محیطی عملکرد پایدار داشته باشد

**در این پروژه، یک محیط آزمایشی ۵×۵ طراحی شده که ویژگی‌های زیر را دارد:**

- موانع ثابت (صخره‌ها) به صورت تصادفی توزیع شده‌اند

- کاوشگر فقط می‌تواند به چهار جهت اصلی (بالا، پایین، چپ، راست) حرکت کند

**محدودیت‌های حرکتی:**

- عدم امکان حرکت مورب

- ممنوعیت عبور از سلول‌های با مقدار ۱

- الزام به رسیدن به نقطه هدف در کوتاه‌ترین زمان

**روش‌شناسی و پیاده‌سازی**

**الف) طراحی الگوریتم:**

**مراحل اصلی الگوریتم BFS پیاده‌سازی شده:**

۱. شروع از نقطه اولیه (۰,۰) و افزودن به صف

۲. بررسی همسایه‌های چهارگانه سلول جاری

۳. افزودن سلول‌های معتبر به صف

۴ . تکرار فرآیند تا رسیدن به نقطه هدف

۵. ثبت مسیر نهایی

**ب) ساختار داده‌ها:**

۱. ماتریس ۵×۵ برای نمایش محیط

**grid = [**

**[0, 0, 0, 1, 0],**

**[1, 1, 0, 1, 0],**

**[0, 0, 0, 1, 0],**

**[0, 1, 1, 1, 0],**

**[0, 0, 0, 0, 0]**

**]**

۲. صف (Queue) برای مدیریت سلول‌های در حال بررسی:

**from collections import deque**

**queue = deque()**

۳. مجموعه (Set) برای ثبت سلول‌های بازدید شده:

**visited = set ()**

**ج) توابع اصلی:**

۱. تابع اعتبارسنجی شبکه:

**def validate\_grid(grid):**

* بررسی ابعاد شبکه

**if len(grid)!= 5 or any(len(row) != 5 for row in grid):**

**return False**

* بررسی مقادیر سلول‌ها

**for row in grid:**

**for cell in row:**

**if cell not in {0, 1}:**

**return False**

**return True**

**۲**. تابع اصلی BFS:

**def bfs\_shortest\_path(grid):**

**# اعتبارسنجی اولیه**

**if not validate\_grid(grid):**

**return "Invalid grid"**

* تعریف جهت‌های ممکن

**directions = [(-1,0), (1,0), (0,-1), (0,1)]**

* مقداردهی اولیه

**queue = deque ([(0, 0, [(0,0)])])**

**visited = set ((0,0))**

* حلقه اصلی الگوریتم

**while queue:**

**row, col, path = queue.popleft()**

* بررسی رسیدن به هدف

**if row == 4 and col == 4:**

**return path**

* بررسی همسایه‌ها

**for dr, dc in directions:**

**new\_row, new\_col = row + dr, col + dc**

**if 0 <= new\_row < 5 and 0 <= new\_col < 5:**

**if grid[new\_row][new\_col] == 0 and (new\_row, new\_col) not in visited:**

**visited.add((new\_row, new\_col))**

**queue.append((new\_row, new\_col, path + [(new\_row, new\_col)]))**

**return "No path found"**

نتایج و تحلیل

**آزمایش‌های انجام شده:**

۱. سناریوی اول (مسیر واضح):

- ورودی:

**grid = [**

**[0,0,0,0,0],**

**[0,1,1,1,0],**

**[0,1,0,1,0],**

**[0,1,0,1,0],**

**[0,0,0,0,0]**

**]**

-خروجی:

**[(0,0), (0,1), (0,2), (0,3), (0,4),**

**(1,4), (2,4), (3,4), (4,4)]**

۲. سناریوی دوم (مسیر پیچیده):

-ورودی:

**grid = [**

**[0,1,0,0,0],**

**[0,1,0,1,0],**

**[0,0,0,1,0],**

**[1,1,1,1,0],**

**[0,0,0,0,0]**

**]**

**- خروجی:**

**[(0,0), (1,0), (2,0), (2,1), (2,2),**

**(1,2), (0,2), (0,3), (0,4), (1,4),**

**(2,4), (3,4), (4,4)]**

**تحلیل عملکرد**

|  |  |
| --- | --- |
| Amount | Metric |
| 0.01 s < | **Running Time** |
| 1 MB < | **Space** |
| 25 max | **Steps** |

**محدودیت‌ها:**

۱ . فقط برای شبکه‌های کوچک مناسب است

۲ . در محیط‌های با ابعاد بزرگ کارایی کمتری دارد

۳ . نیاز به توسعه برای موانع متحرک

**نتیجه‌گیری**

سیستم توسعه داده شده نشان‌دهنده یک راه‌حل پایدار و کارآمد برای مسئله مسیریابی کاوشگرهای مریخی در محیط‌های محدود است. الگوریتم BFS با وجود سادگی، عملکرد مناسبی در یافتن کوتاه‌ترین مسیر دارد.

برای بهبود پیچیدگی زمانی و حافظه میتوان از الگوریتم های دیگر نیز بهره برد . مانند A\*

همچنین با تغییراتی میتوان کد را آماده دریافت ورودی های بیشتری نمود

زیرا استفاده از کد فعلی برای داده های بزرگ و حجیم بهینه نیست پس

میتوان با تغییراتی کد فعلی را بهبود داد.

**ضمیمه‌ها**

**کد کامل پروژه:**

**from collections import deque**

**def validate\_grid(grid):**

**if len(grid) != 5 or any(len(row) != 5 for row in grid):**

**return False**

**for row in grid:**

**for cell in row:**

**if cell not in {0, 1}:**

**return False**

**return True**

**def bfs\_shortest\_path(grid):**

**if not validate\_grid(grid):**

**return "Invalid grid format"**

**if grid[0][0] == 1 or grid[4][4] == 1:**

**return "Start or end position is blocked"**

**directions = [(-1,0), (1,0), (0,-1), (0,1)]**

**queue = deque([(0, 0, [(0,0)])])**

**visited = set((0,0))**

**while queue:**

**row, col, path = queue.popleft()**

**if row == 4 and col == 4:**

**return path**

**for dr, dc in directions:**

**new\_row, new\_col = row + dr, col + dc**

**if 0 <= new\_row < 5 and 0 <= new\_col < 5:**

**if grid[new\_row][new\_col] == 0 and (new\_row, new\_col) not in visited:**

**visited.add((new\_row, new\_col))**

**queue.append((new\_row, new\_col, path + [(new\_row, new\_col)]))**

**return "No valid path exists"**

**sample\_grid = [**

**[0,0,0,1,0],**

**[1,1,0,1,0],**

**[0,0,0,1,0],**

**[0,1,1,1,0],**

**[0,0,0,0,0]**

**]**

**print(bfs\_shortest\_path(sample\_grid))**

**منابع و ماخذ**

بررسی مجدد الگوریتم : BFS

<https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/>

<https://github.com/omonimus1/geeks-for-geeks-solutions/blob/master/c++/bfs-graph.cpp>

یادگیری بیشتر و مرور زبان پایتون :

<https://www.w3schools.com/python/ref_func_any.asp>

<https://www.w3schools.com/python/python_lists.asp>

<https://www.w3schools.com/python/python_tuples.asp>

<https://www.w3schools.com/python/python_sets.asp>

<https://www.w3schools.com/python/python_functions.asp>

<https://www.w3schools.com/python/python_dsa_queues.asp>

<https://www.w3schools.com/python/python_dsa_lists.asp>