Breadth First Search

پیدا کردن کوتاه‌ترین مسیر از start به goal ، در این حالت برای بررسی مسیر رسیدن به هدف ، مسیر ها را بدون وزن در نظر میگیرد.

از صف استفاده می‌کند چون BFS به ترتیب سطح کار می‌کند.

- ازvisited برای نگه‌داری بلاک های دیده شده استفاده می‌شود.
- از checked_list فقط برای شمارش خانه‌هایی که بررسی شدند استفاده شده.

چگونه کار میکند :

1. نقطه شروع را به صف اضافه می‌کند.
2. تا زمانی که صف خالی نیست:
    - اولین عنصر صف را بیرون می‌آورد
    - اگر این نقطه هدف بود ←حلقه تمام.
    - در غیر اینصورت، همه همسایه‌ها را از neighbors_fn می‌گیرد.
    - اگر همسایه قبلاً بازدید نشده بود:
        - آن را به صف اضافه می‌کند.
        - در visitedمی‌نویسد از کجا آمده.
3. وقتی هدف پیدا شد، از طریق visitedمسیر را از هدف به شروع برمی‌گرداند و برعکس می‌کند تا مسیر مستقیم شود.

DFS — Depth First Search

پیدا کردن مسیر از start به goal با جستجوی عمقی تا جایی که ممکن است می‌رود

چگونه کار میکند :

- از stack پشته استفاده می‌کند
- از visited برای جلوگیری از بازدید تکراری استفاده میکند.
- نقطه شروع را در stack قرار می‌دهد.

1. تا وقتی stack خالی نیست:
    - آخرین عنصر stack را بیرون می‌آورد
    - اگر هدف پیدا شد ← تمام.
    - همه همسایه‌های گره را می‌گیرد.
    - هر همسایه‌ای که دیده نشده باشد ← در stack قرار می‌دهد.
2. بعد از رسیدن به هدف، مسیر را مانند BFS بازسازی می‌کند.

UCS — Uniform Cost Search

پیدا کردن کم‌هزینه‌ترین مسیر از start تا goal در گرافی که بلاک ها وزن دارند.

- از صف اولویت‌دار برای نگه‌داری گره‌ها با کمترین هزینه استفاده می‌کند.
- heapشامل زوج‌هایی از (هزینه کل، موقعیت) است.
- visited[node] = (parent, total_cost) است .

1. نقطه شروع را با هزینه 0 در heap قرار می‌دهد.
2. تا زمانی که heap خالی نیست:
   - گره با کمترین هزینه را از heap بیرون می‌آورد.
   - اگر گره همان goal است ← مسیر پیدا شد.
   - در غیر اینصورت، همسایه‌های گره را می‌گیرد.
   - هزینه جدید هر همسایه = وزن بلاک + هزینه فعلی
   - اگر همسایه هنوز دیده نشده ← آن را در heap می‌گذارد. (در این سوال نیاز به بررسی اینکه هزینه جدید کمتر از هزینه رسیدن به بلاک در دست بررسی نیست زیرا قطعا به دلیل استفاده صف اولویت دار و اینکه همه بلاک ها برخلاف گراف ها که ممکن است یال ها همه مقدار متفاوتی برای رسیدن به یک نود داشته باشند , بلاک ها هزینه ثابت دارند در صورت بررسی شدن قبلی هزینه فعلی بزرگتر از هزینه قبلی است)
3. وقتی goal پیدا شد، از visitedمسیر را بازسازی می‌کند.

A* Search

پیدا کردن کوتاه‌ترین مسیر مانند UCS، اما با کمک تابع heuristic برای تخمین فاصله تا هدف و سریع‌تر شدن جستجو برای استفاده میکند.

- از heapq مانند UCS استفاده می‌کند.
- Heuristic یا "manhattan"یا "euclidean"است.
- visited[node] = (parent, cost_so_far) است.

2. گره شروع را با اولویت 0 در heap قرار می‌دهد.
3. تا وقتی heap خالی نیست:
   - گره با کمترین (cost + heuristic)را بیرون می‌آورد.
   - اگر این گره goal است ← مسیر پیدا شد.
   - در غیر اینصورت، همه همسایه‌ها را بررسی می‌کند:
     - هزینه واقعی تا الان=g
     - h = heuristic
     - f = g + h
   - اگر همسایه هنوز دیده نشده ← آن را در heap می‌گذارد. (در این سوال نیاز به بررسی اینکه هزینه جدید کمتر از هزینه رسیدن به بلاک در دست بررسی نیست زیرا قطعا به دلیل استفاده صف اولویت دار و اینکه همه بلاک ها برخلاف گراف ها که ممکن است یال ها همه مقدار متفاوتی برای رسیدن به یک نود داشته باشند , بلاک ها هزینه ثابت دارند در صورت بررسی شدن قبلی هزینه فعلی بزرگتر از هزینه قبلی است)
4. مسیر نهایی با بازگشت از visitedساخته می‌شود.

بررسی زمان و بلاک و هزینه :

: BFS

شماره مسیر	زمان	بلاک	هزینه
1	0.000650	69	10
2	0.000730	154	10
3	0.001033	149	11
4	0.000660	82	10
	0.000768	113.5	10.25

: DFS

شماره مسیر	زمان	بلاک	هزینه
1	0.000951	164	86
2	0.002378	180	44
3	0.000891	138	29
4	0.000661	134	72
	0.001220	154	57.75

: UCS

شماره مسیر	زمان	بلاک	هزینه
1	0.000581	65	10
2	0.000868	154	12
3	0.001017	117	11
4	0.000438	69	10
	0.000726	101.25	10.75

: A* (Manhattan)

شماره مسیر	زمان	بلاک	هزینه
1	0.000528	30	10
2	0.000747	59	12

11	42	0.000688	3
10	22	0.000310	4
10.75	38.25	0.000568	

A* (Euclidean) :

هزینه	بلاک	زمان	شماره مسیر
10	36	0.000472	1
12	68	0.000780	2
11	44	0.000352	3
10	27	0.000556	4
10.75	43.75	0.000540	