بسمه تعالی

گزارش پروژه‌ی اول هوش مصنوعی

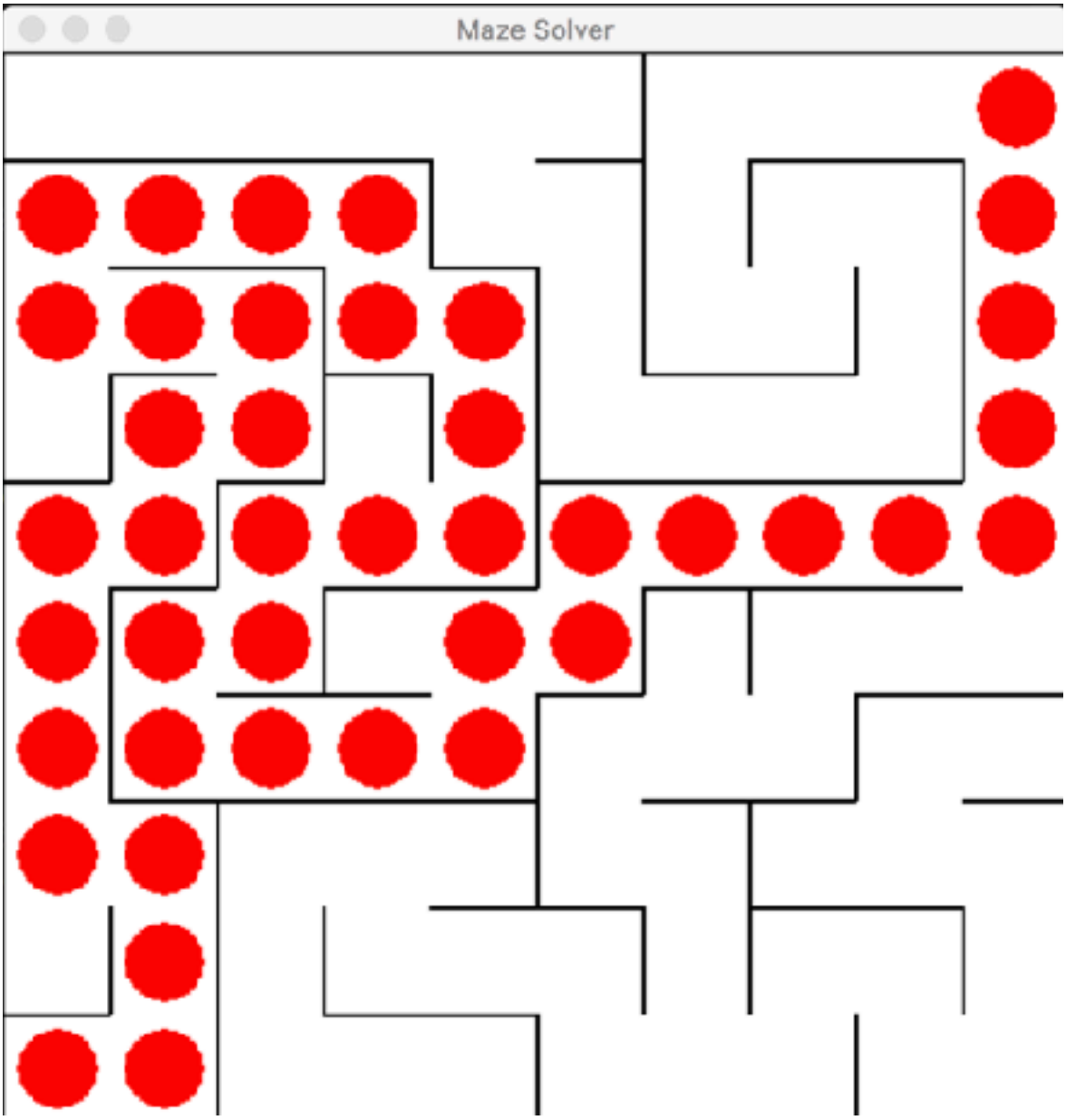
روزبه بستان‌دوست **۸۱۰۱۹۴۲۷۷**

پویا نقوی **۸۱۰۱۹۴۴۱۷**

در این پروژه باید توسط ۴ الگوریتم، BFS، DFS، Iterative DFS و A* از نقطه‌ی شروع یک ماز به نقطه‌ی پایانی آن حرکت کنیم. در ادامه به توضیح مختصر هریک می‌پردازیم.

BFS:

در این الگوریتم به پیمایش سطحی گراف می‌پردازیم. به این نحو که از نقطه‌ی ابتدایی ماز شروع کرده و میان تمامی همسایه‌های آن نقطه اگر دیواری بین آن و همسایه وجود نداشت، آن همسایه را به انتهای صفی اضافه می‌کنیم. اکنون یک عنصر از ابتدای لیست انتخاب کرده و این کار را برای آن تکرار می‌کنیم. به این نحو به پیمایش سطحی گراف می‌پردازیم. برای اینکه در نهایت مسیر طی شده از مبدا تا مقصد را تعیین کنیم، parent هر یک عناصری که به صف ذکر شده اضافه می‌کنیم را توسط هش نگه می‌داریم.
در نهایت توسط هش موجود از نقطه‌ی ابتدایی به نقطه‌ی انتهایی منتقل می‌شویم.
در شکل ۱ خروجی کد پس از اجرای الگوریتم را مشاهده می‌کنیم.

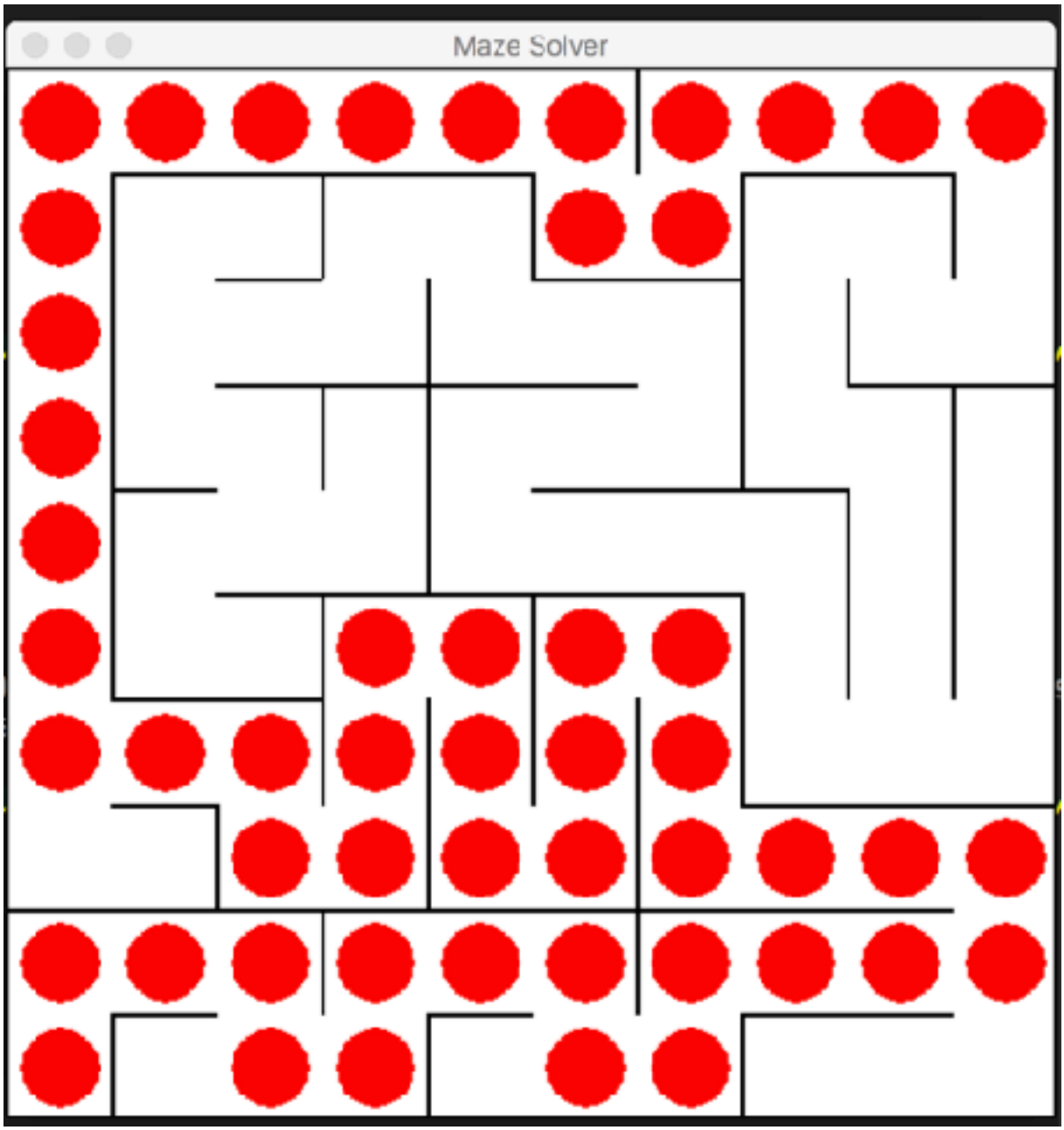


شکل ۱(الگوریتم BFS)

:DFS

در این الگوریتم به پیمایش عمقی گراف می‌پردازیم. به این نحو که از نقطه‌ی ابتدایی ماز شروع کرده و میان تمامی همسایه‌های آن نقطه اگر دیواری بین آن و همسایه وجود نداشت، آن همسایه را به انتهای صفی اضافه می‌کنیم. برخلاف BFS که از ابتدای صف یکی را جدا می‌کردیم، اکنون از انتهای صف یکی را انتخاب کرده و همین روند را با همسایه‌های آن را تکرار می‌کنیم. توجه داشته باشیم که برای اضافه کردن هر همسایه به لیست ابتدا چک می‌کنیم اگر این خانه قبلا در صف قرار داشت، آن را حذف و سپس دوباره آن را به انتهای صف اضافه می‌کنیم. چرا که پیمایش آن باید سریع‌تر انجام شود. به این شکل در هر مرحله به عمق گراف خود حرکت می‌کنیم. برای اینکه در نهایت مسیر طی

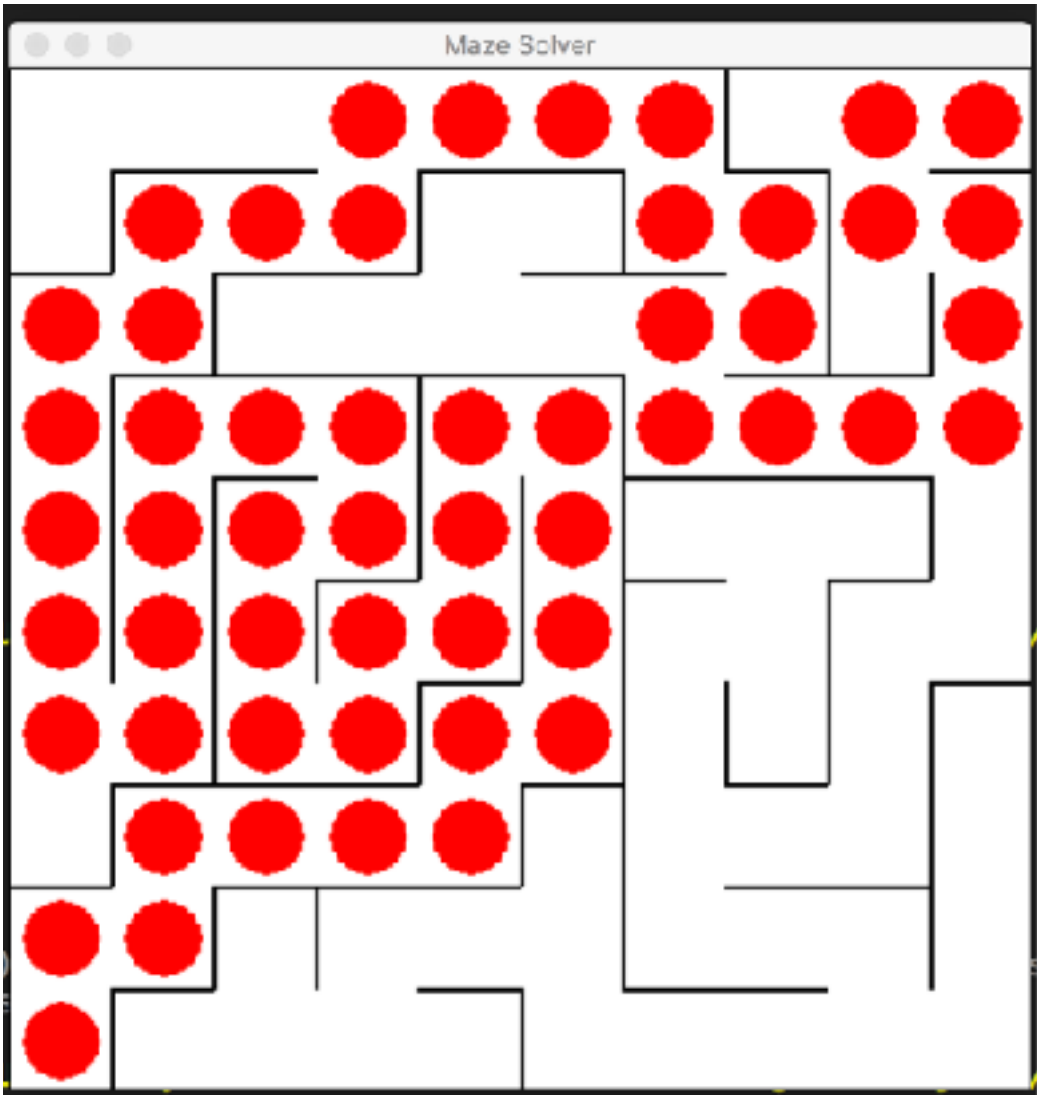
شده از مبدا تا مقصد را تعیین کنیم، parent هر یک عناصری که به صف ذکر شده اضافه می‌کنیم را توسط هش نگه می‌داریم.
در نهایت توسط هش موجود از نقطه‌ی ابتدایی به نقطه‌ی انتهایی منتقل می‌شویم.
در شکل ۲ خروجی کد پس از اجرای الگوریتم را مشاهده می‌کنیم.



شکل ۲ (الگوریتم DFS)

Iterative DFS:

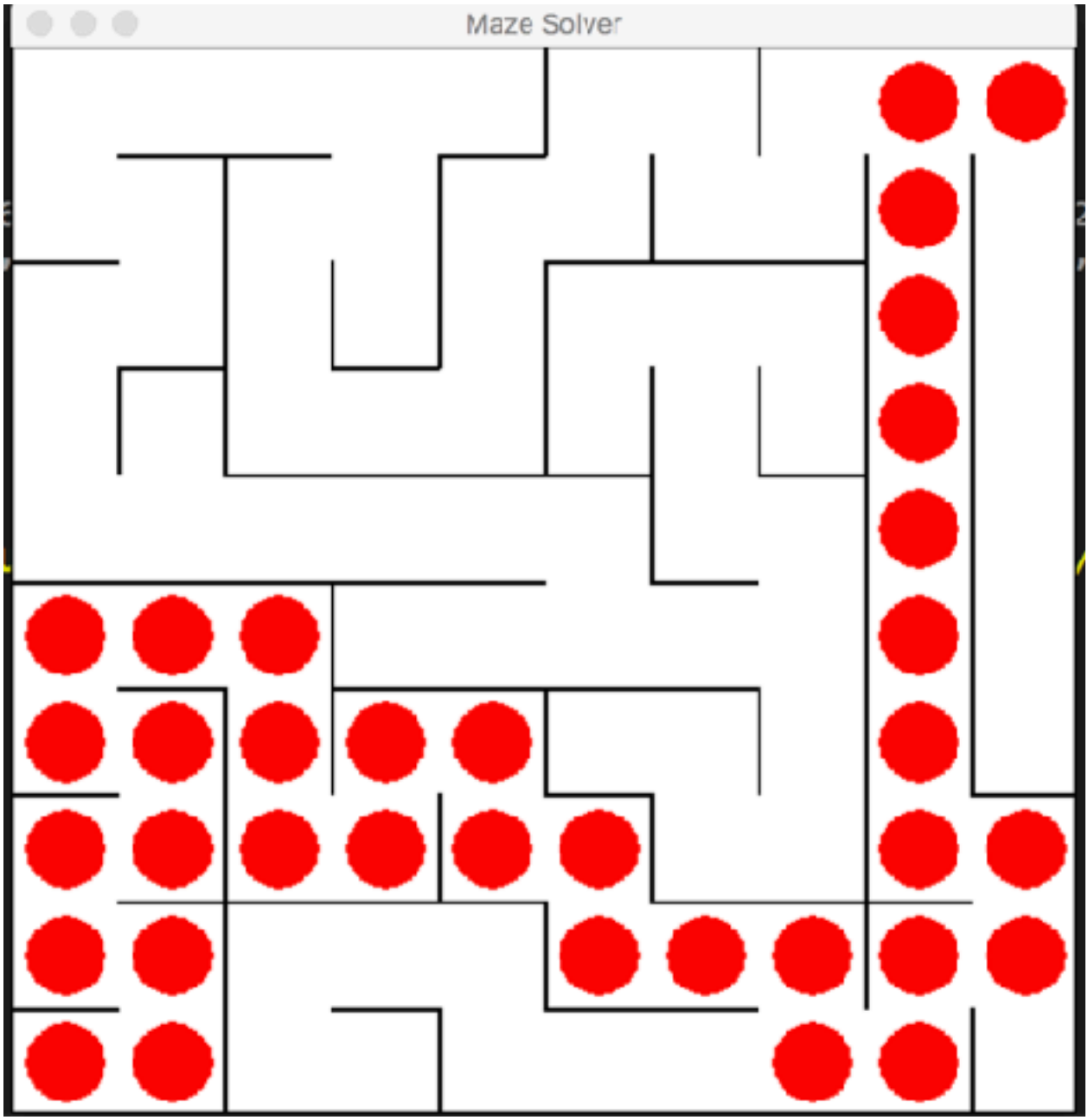
در این الگوریتم ما با حرکت روی یک for از ۱ تا حاصل‌ضرب تعداد ردیف‌ها در تعداد ستون‌ها، در هر مرحله تابع dls_solver را فراخوانی می‌کنیم به نحوی که مقدار limit آن برابر متغیر for باشد. در تابع dls_solver یک DFS دارای لیمیت نوشته شده است. این تابع مشخص می‌کند که اگر پیمایش عمقی گراف را فقط تا عمق مشخصی که در ورودی دریافت می‌کند، انجام دهد، آیا به مقصد می‌رسد یا خیر. مثلا اگر به عنوان ورودی مقدار limit = 1 به تابع داده شود، این تابع فقط الگوریتم DFS را برای عمق ۱ انجام می‌دهد. (تنها فرزندان خود را بررسی می‌کند)
پس به طور کلی در این الگوریتم از کمترین عمق شروع کرده و به طور متناوب الگوریتم DFS دارای لیمیت را فراخوانی می‌کنیم تا در نهایت در ازای یکی از لیمیت‌ها، خانه‌ی مقصد در جواب نهایی ما قرار بگیرد. لازم به ذکر است که سایر موارد الگوریتم همانند الگوریتم DFS پیاده‌سازی شده است و تنها تفاوت آن این است که اگر مقدار لیمیت از صفر بزرگتر بود فرزندان را به صف اضافه کند.
در شکل ۳ خروجی کد پس از اجرای الگوریتم را مشاهده می‌کنیم.



شکل۳ (الگوریتم Iterative DFS)

A*:

در این الگوریتم ما ابتدا به هر خانه مقداری نسبت می‌دهیم که برابر است با تخمین ما از هزینه‌ای که باید متحمل شویم تا از آن خانه به خانه‌ی مقصد برسیم. به این مقدار h(n) گفته می‌شود.
سپس به هر خانه مقداری نسبت می‌دهیم که برابر است با هزینه‌ای که تا اینجا متحمل شدیم تا به این خانه برسیم. به این مقدار g(n) گفته می‌شود.
ما برای تعیین h(n) فاصله‌ی مستقیم تا خانه‌ی مقصد را به عنوان تابع تخمین انتخاب کردیم که از رابطه‌ی رادیکال (y2-y1)^2 + (x2-x1)^2 بدست می‌آید. چرا که هرچه این مقدار کمتر باشد، یعنی ما به جواب نزدیک‌تر هستیم و یعنی این خانه ایده‌آل‌تر است.
برای تعیین g(n) تعداد خانه‌های طی شده برای رسیدن به این خانه را معیار قرار دادیم.
پس از تعیین این تو تابع اکنون با پیدا کردن مینمم مقدار h(n) + g(n) در هر مرحله، خانه‌ی بعدی خود را انتخاب می‌کنیم.
پس با این روال یک BFS کلی داریم که به همان شکل تمامی همسایه‌ها را به یک صف اضافه می‌کند و میان تمامی اعضای صف، مینمم مقدار ذکر شده را پیدا کرده و آن خانه را بسط می‌دهیم.
در انتها همانند الگوریتم BFS توسط تابع هش، مسیر طی شده را به وسیله‌ی parentهای ذخیره کرده، تعیین می‌کنیم.
در شکل ۴ خروجی کد پس از اجرای این الگوریتم را مشاهده می‌کنیم.



شکل ۴ (الگوریتم A^*)