به نام خدا

دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر و فناوری اطلاعات

گزارش پروژه ۱

سید نوید کرمی‌نژاد

۹۴۳۱۰۷۰

آذر ۹۶

شرح پروژه:

در این پروژه، مجموعه‌ای از الگوریتم‌های جستجوی کلاسیک که فهرست آن‌ها در زیر آورده شده‌است پیاده‌سازی و از آن‌ها برای حل چند مسئله جستجوی مختلف که در قالب یک واسط مشخص پیاده‌سازی می‌شوند، استفاده شده است.

الگوریتم‌ها:

الگوریتم‌هایی که باید پیاده‌سازی شوند، عبارت‌اند از:

- سطح اول
- عمق اول(در سه حالت نامحدود، با عمق محدود و با افزایش تدریجی عمق)
- هزینه یکنواخت
- دو جهته
- A*

که در دو حالت درختی و گرافی پیاده‌سازی شده‌اند.

مسائل:

به ازای هر یک از مسائل باید یک داده ساختار State برای نگهداری حالت مسئله و یک کلاس Problem تعیین شود که در آن توابع لازم برای تعریف مسئله (تابع حالت اولیه، عمل‌های ممکن در هر حالت، نتیجه عمل، آزمایش هدف، هزینه مسیر و گام، و تابع شهودی مورد استفاده) پیاده‌سازی شده‌باشند.

مسئله پارچ آب: در این مسئله دو پارچ با سایزهای ۳و۴ لیتر و یک شیر آب که به مقدار نامحدود آب دارد، داده‌شده‌است. هدف این است که با جابه‌جایی آب بین پارچ‌ها و پر و خالی کردن آن‌ها، در پارچ ۴ لیتری به اندازه ۲ لیتر آب موجود باشد.

حالت (State) مسئله: نگهداری حجم آب موجود در هر پارچ

حالت اولیه : هر دو پارچ خالی هستند

حالت هدف : پارچ ۴ لیتری، ۲ لیتر آب داشته باشد

- نتایج حاصل از الگوریتم عمق اول (گرافی)

```
(0,0)
(0,3)
(3,0)
(3,3)
(4,2)
(0,2)
(2,0)
NODE NUMBER EXIST: 7.0
PATH COST: 6.0
MAX MEMORY USAGE: 8
EXPANDED NODES: 6
VISITED NODES: 6
RUN TIME:12 MS
```

- نتایج حاصل از الگوریتم دوجهته (گره مشترک: (۳و۳))

```
FROM ROOT
(0,0)
(0,3)
(3,0)
(3,3)
FROM Goal
(2,0)
(0,2)
(4,2)
(3,3)
NODE NUMBER EXIST: 7.0
PATH COST: 6.0
MAX MEMORY USAGE: 26
EXPANDED NODES: 77
VISITED NODES: 77
RUN TIME: 12 MS
```

با توجه به نتایج فوق می‌توان بیان کرد سرعت اجرای هر دو الگوریتم تقریبا یکسان بوده اما حافظه مصرفی در الگوریتم دوجهته بیشتر است. در واقع می‌توان گفت این الگوریتم برای حل این مسئله مناسب نیست.

مسئله پازل ۸ تایی: این مسئله یکی از معروف‌ترین مسائل جستجو است. در این مسئله یک ماتریس ۳*۳ به عنوان ورودی داده می‌شود که تنها خانه ۰ آن با خانه مجاورش می‌تواند جابه‌جا شود. در نهایت با حل این مسئله باید به ماتریس زیر برسیم

۰	۱	۲
۳	۴	۵
۶	۷	۸

حالت (State) مسئله: نگهداری ماتریسی از وضعیت فعلی اعداد و جایگاه خانه‌ای که عدد ۰ در آن قرار دارد

حالت اولیه : ورودی دلخواه کاربر

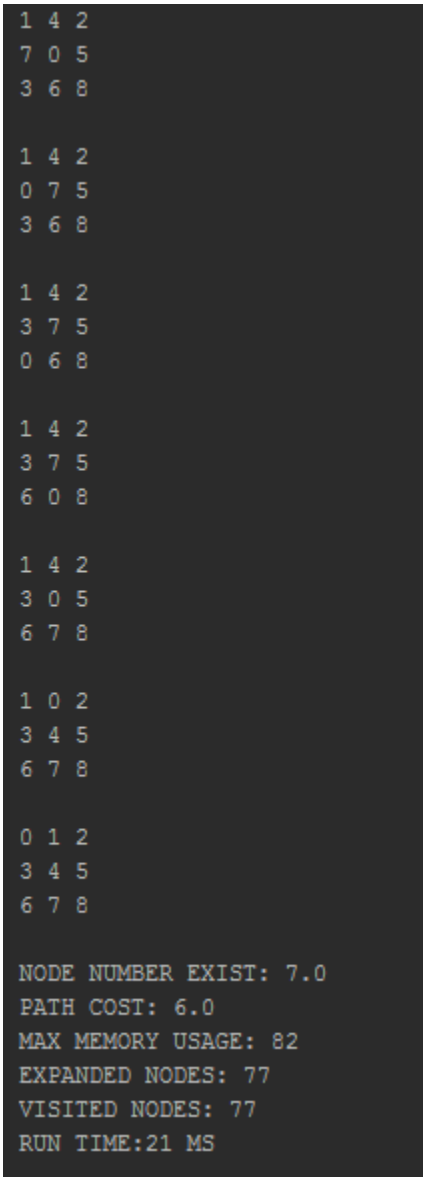
حالت هدف : رسیدن وضعیت ماتریس به حالت بالا

- نتایج حاصل از الگوریتم سطح اول (درختی)

*توجه شود که به ازای برخی حالات اولیه، زمان اجرای الگوریتم بسیار زیاد خواهد بود.

```
1 4 2
7 0 5
3 6 8

1 4 2
0 7 5
3 6 8

1 4 2
3 7 5
0 6 8

1 4 2
3 7 5
6 0 8

1 4 2
3 0 5
6 7 8

1 0 2
3 4 5
6 7 8

0 1 2
3 4 5
6 7 8

NODE NUMBER EXIST: 7.0
PATH COST: 6.0
MAX MEMORY USAGE: 560
EXPANDED NODES: 298
VISITED NODES: 298
RUN TIME:33 MS
```

- نتایج حاصل از الگوریتم عمق اول (با عمق محدود ۸)



- نتایج حاصل از الگوریتم A* با تابع شهودی مجموع فواصل مستقیم خانه هر عدد با خانه‌ای که باید حضور داشته باشد.

```
1 4 2
7 0 5
3 6 8

1 4 2
0 7 5
3 6 8

1 4 2
3 7 5
0 6 8

1 4 2
3 7 5
6 0 8

1 4 2
3 0 5
6 7 8

1 0 2
3 4 5
6 7 8

0 1 2
3 4 5
6 7 8

NODE NUMBER EXIST: 7.0
PATH COST: 6.0
MAX MEMORY USAGE: 21
EXPANDED NODES: 10
VISITED NODES: 10
RUN TIME:16 MS
```

بر اساس نتایج فوق به وضوح مشخص می‌شود که الگوریتم A* از تمامی جهات بهتر عمل کرده‌است (حافظه مصرفی به مراتب کمتر و زمان اجرای کمتر) با اینکه تمامی الگوریتم‌ها یک مسیر یکسان را به عنوان مسیر نهایی مشخص کرده‌اند.

پس از آن الگوریتم عمق اول محدود نتایج بهتری نسبت به سطح اول دارد. پس می‌توان گفت مناسب‌ترین الگوریتم برای حل این مسئله از میان الگوریتم‌های انتخاب شده A* است.

مسئله مسیریابی ربات: در این مسئله یک ربات در یک محیط n*m خانه‌ای، قرار است از نقطه شروع (0,0) به نقطه هدف (n-1,m-1) برود. ربات در هر

حرکت می‌تواند به یکی از خانه‌های مجاور خود (بالا، پایین، چپ و راست) حرکت کند. ربات از خانه‌هایی که با مقدار صفر مشخص شده‌اند نم‌تواند عبور کند.

حالت (State) مسئله: نگهداری ماتریسی از وضعیت فعلی ربات

حالت اولیه : ورودی دلخواه کاربر

حالت هدف : رسیدن ربات به خانه (n-1,m-1)

- نتایج حاصل از الگوریتم هزینه یکنواخت (هزینه حرکت از هر خانه به خانه مجاور آن را ۱ در نظر می‌گیریم)

```
2 1 1 0
0 0 1 1
1 1 1 0
0 1 0 1
1 1 0 0
1 1 1 1

1 2 1 0
0 0 1 1
1 1 1 0
0 1 0 1
1 1 0 0
1 1 1 1

1 1 2 0
0 0 1 1
1 1 1 0
0 1 0 1
1 1 0 0
1 1 1 1

1 1 1 0
0 0 2 1
1 1 1 0
0 1 0 1
1 1 0 0
1 1 1 1

1 1 1 0
0 0 1 1
1 1 2 0
0 1 0 1
1 1 0 0
1 1 1 1
```

```
1 1 1 0
0 0 1 1
1 2 1 0
0 1 0 1
1 1 0 0
1 1 1 1

1 1 1 0
0 0 1 1
1 1 1 0
0 2 0 1
1 1 0 0
1 1 1 1

1 1 1 0
0 0 1 1
1 1 1 0
0 1 0 1
1 2 0 0
1 1 1 1

1 1 1 0
0 0 1 1
1 1 1 0
0 1 0 1
1 1 0 0
1 2 1 1

1 1 1 0
0 0 1 1
1 1 1 0
0 1 0 1
1 1 0 0
1 1 2 1
```

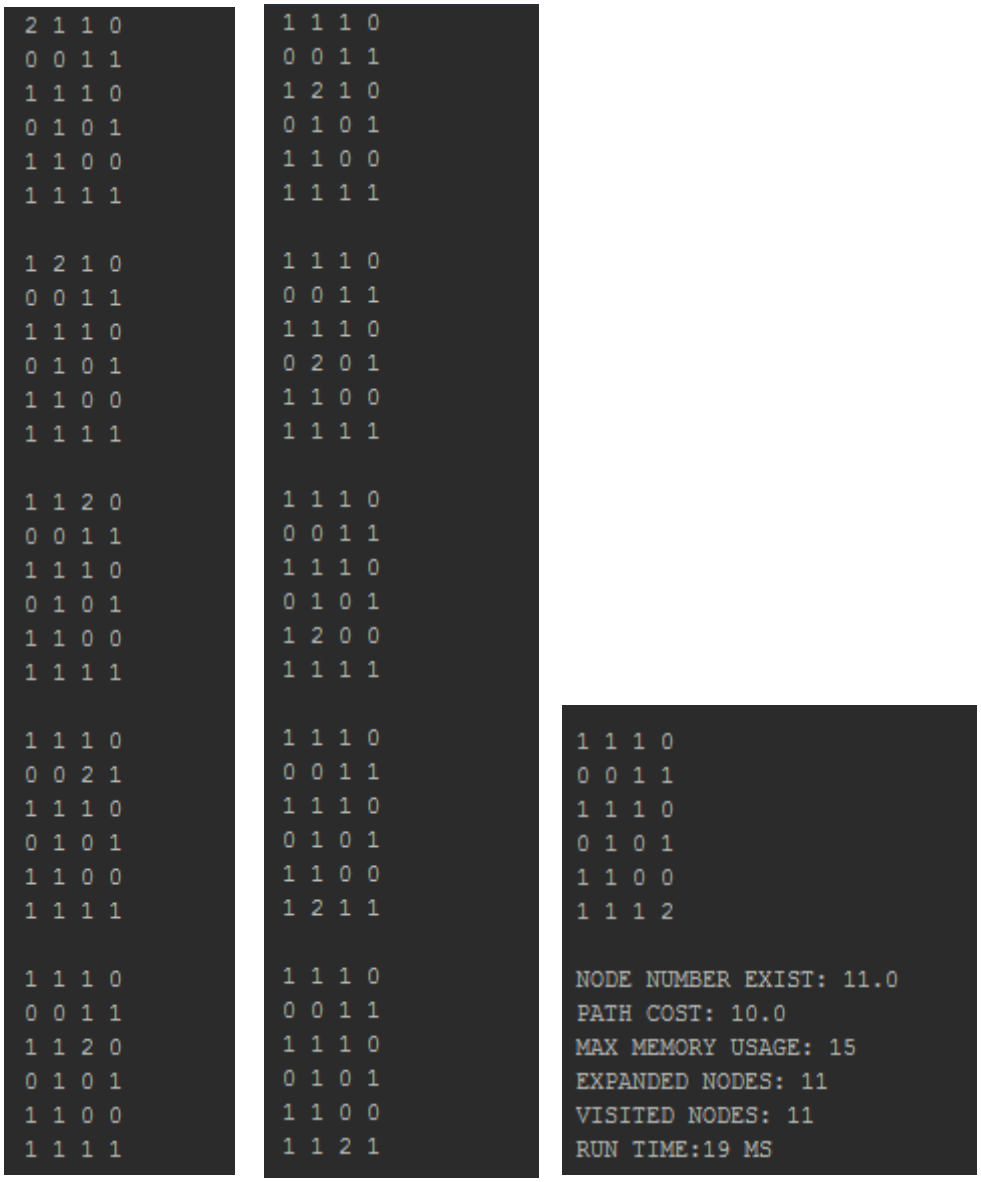
```
1 1 1 0
0 0 1 1
1 1 1 0
0 1 0 1
1 1 0 0
1 1 1 2

NODE NUMBER EXIST: 11.0
PATH COST: 10.0
MAX MEMORY USAGE: 15
EXPANDED NODES: 14
VISITED NODES: 14
RUN TIME:21 MS
```

- نتایج حاصل از الگوریتم دوجهته

```
FROM ROOT
2 1 1 0
0 0 1 1
1 1 1 0
0 1 0 1
1 1 0 0
1 1 1 1

1 2 1 0
0 0 1 1
1 1 1 0
0 1 0 1
1 1 0 0
1 1 1 1

1 1 2 0
0 0 1 1
1 1 1 0
0 1 0 1
1 1 0 0
1 1 1 1

1 1 1 0
0 0 2 1
1 1 1 0
0 1 0 1
1 1 0 0
1 1 1 1

1 1 1 0
0 0 1 1
1 1 2 0
0 1 0 1
1 1 0 0
1 1 1 1
```

```
1 1 1 0
0 0 1 1
1 2 1 0
0 1 0 1
1 1 0 0
1 1 1 1

FROM Goal
1 1 1 0
0 0 1 1
1 1 1 0
0 1 0 1
1 1 0 0
1 1 1 2

1 1 1 0
0 0 1 1
1 1 1 0
0 1 0 1
1 1 0 0
1 1 2 1

1 1 1 0
0 0 1 1
1 1 1 0
0 1 0 1
1 1 0 0
1 2 1 1

1 1 1 0
0 0 1 1
1 1 1 0
0 1 0 1
1 2 0 0
1 1 1 1

1 1 1 0
0 0 1 1
1 1 1 0
0 2 0 1
1 1 0 0
1 1 1 1

1 1 1 0
0 0 1 1
1 2 1 0
0 1 0 1
1 1 0 0
1 1 1 1

NODE NUMBER EXIST: 11.0
PATH COST: 10.0
MAX MEMORY USAGE: 16
EXPANDED NODES: 66
VISITED NODES: 66
RUN TIME: 24 MS
```

- نتایج حاصل از الگوریتم A* با تابع شهودی فاصله منهتن



بر اساس نتایج مشاهده شده در بالا باز هم می‌توان گفت الگوریتم A* هم از لحاظ مصرف حافظه و هم سرعت اجرا از سایر الگوریتم‌های بررسی شده بهتر است. در بین دو الگوریتم هزینه یکنواخت و دوجهته، هزینه یکنواخت حافظه کمتری مصرف می‌کند اما زمان اجرای الگوریتم آن بیشتر است.