به نام خدا

هوش مصنوعی

تمرین کامپیوتری شماره 1 : Informed and Uniformed Search

امیرحسین کهربائیان – 810199478

شرح پروژه :

در این پروژه باید با استفاده از الگوریتم های مختلف سرچ مسئله DZ Day را حل کنیم. مسئله به این صورت است که در یک گراف تعدادی مرید قرار دارند که هر مرید تعدادی دستور پخت نیاز دارد که در رئوس مختلف قرار دارد. ما باید در کوتاه ترین زمان ممکن دستور پخت های هر مرید را به آن برسانیم. همچنین در گراف ممکن است بعضی رئوس صعب العبور باشند، به این معنی که اگر قبلا از آن راس i بار رد شده باشیم، برای رد شدن مجدد از آن راس باید i ثانیه فکر کنیم و سپس می توانیم از آن راس عبور کنیم. جا به جایی بین هر دو راس یک ثانیه طول می کشد. در ابتدا سید در راس شروع قرار می گیرد و پس از عبور از راس حاوی دستور پخت آن را حفظ می کند و سعی بر رساندن دستور به مرید خودش دارد.

مدل کردن مسئله :

برای حل این مسئله در ابتدا دو کلاس Graph و State را تعریف می کنیم. توضیحات و اطلاعات هر کلاس در ادامه ذکر می شود.

هر ***state*** شامل موارد زیر است :

- یک شی از کلاس گراف که شامل موارد زیر می شود :
  - راس فعلی که سید در آن حضور دارد.
  - تعداد مرید ها و دستور پخت هایی که نیاز دارند.
  - دستور پخت هایی که سید آن ها را حفظ کرده و برعکس.
  - رئوس صعب العبور و مرتبه عبور از آن ها.
- لیست مرید هایی که دستور پخت به آن ها رسیده است.
- رئوسی که سید تا این استیت عبور کرده است.
- زمانی که تا رسیدن به این استیت سپری شده است.
- میزان زمانی که سید در راس صعب العبور در حال تفکر سپری کرده است.

Initial State:

همان شرایط اولیه ای که به عنوان ورودی دریافت می شود.

Goal State:

در استیتی که تمامی دستور پخت های هر مرید به خودش تحویل داده شده باشد، به استیت هدف رسیده ایم.

Actions :

- سید می تواند از هر راس به رئوس همسایه آن راس برود و اگر راس صعب العبور بود باید به میزانی که قبلا از آن راس عبور کرده است توقف کند.
- اگر سید در راسی باشد که در آن دستور پخت وجود داشته باشد بدون مکث آن دستور پخت را حفظ می کند.
- اگر سید به راسی برسد که در آن مریدی وجود داشته باشد، در صورتی که تمام دستور پخت های مرید را حفظ کرده باشد، تمام دستور پخت ها را به مرید بدون مکث تحویل می دهد.

Transition Model :

- سید با رفتن به هر راس دیگر، تمام دستور پخت هایی که حفظ کرده است را فراموش نمی کند و آن ها را به راس دیگر نیز منتقل می کند.
- با حرکت به سمت رئوس همسایه، به آن راس منتقل شده و مکان فعلی سید آپدیت می شود .

Cost :

- انتقال بین دو راس عادی 1 ثانیه زمان میبرد.
- اگر در راس صعب العبور قرار داشتیم و قبلا i بار از این راس عبور کرده باشیم، ابتدا i ثانیه صبر می کنیم و سپس طی یک ثانیه به یکی از رئوس همسایه منتقل شویم.

لازم به ذکر است که کلاس Graph عملیاتی مانند خواندن گراف اولیه، آپدیت کردن دستور پخت های حفظ شده و مکان فعلی سید و مرتبه عبور از رئوس صعب العبور و ... می تواند انجام دهد.

حال باید به بررسی الگوریتم های BFS و IDS و A^* و Weighted A^* بپردازیم و پس از پیاده سازی هر الگوریتم نتایج مربوط به هر تست کیس را ارائه دهیم.

تست کیس های مورد بررسی :

سه تست کیس ابتدایی یعنی in1.txt و in2.txt و in3.txt ، سه تست کیس ایزی هستند که به نام {testcases (easy)} در سایت قرار داده شده است. In4.txt و in5.txt دو تست کیس سنگین تری هستند که در تست کیس های اولیه در سایت قرار داده شده بود. In6.txt و in7.txt نیز دو تست کیس جدید هستند که به طور دستی آماده شده اند .

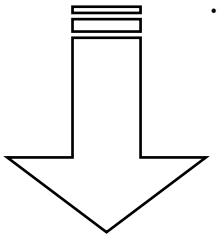
حال برای هر تست کیس، هر الگوریتم را سه بار اجرا می کنیم و میانگین زمان اجرا ، پاسخ مسئله ، مسیر طی شده و تعداد استیت های دیده شده را به دست می آوریم. به کمک قطعه کد زیر این کار را انجام میدهیم، دقت کنید که به جای ALGORITHM() ، الگوریتم های پیاده سازی شده مانند BFS() و ... را جایگزین می کنیم.

```
g = Graph()
g.get_graph()

initial_state = State(graph=g, path = [g.cur_node+1])

ExecutionTimeList = []
for i in range(3) :
start = time.time()
path, cost_of_path, visit_states = ALGORITHM(initial_state)
end = time.time()
ExecutionTimeList.append(end - start)

print(f"Path : {path}\nCost Of Path : {cost_of_path}\
\nExecution Time : {sum(ExecutionTimeList)/3}\nSeen States : {visit_states}")
```

:BFS

این الگوریتم uniformed است و راس ها را بر اساس ارتفاع آنان پیمایش می کند. زمانی که تمام رئوس یک ارتفاع پیمایش شد به پیمایش رئوس ارتفاع بعدی می رود. از list به عنوان یک صف استفاده کرده ایم و رئوس دیده شده را ذخیره می کنیم تا از یک استیت چند بار بازدید نشود.

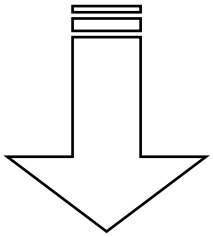
این الگوریتم کامل است و جواب بهینه را می یابد و نسبت به IDS سریع تر است ولی نسبت به الگوریتم A* کند تر است و به حافظه و زمان بیشتری نیاز دارد.



```
amirhossein@amirhossein-u:~/Desktop/CA1$ python3 Search.py < in1.txt
Path : [1, 3, 4, 5, 7, 10, 11, 9, 8]
Cost Of Path : 8
Execution Time : 0.008672714233398438
Seen States : 35
amirhossein@amirhossein-u:~/Desktop/CA1$ python3 Search.py < in2.txt
Path : [9, 10, 9, 4, 12, 3, 7, 5, 8]
Cost Of Path : 8
Execution Time : 0.029268900553385418
Seen States : 90
amirhossein@amirhossein-u:~/Desktop/CA1$ python3 Search.py < in3.txt
Path : [13, 11, 10, 3, 2, 6, 12, 5, 9, 4, 1, 13, 11, 10]
Cost Of Path : 13
Execution Time : 0.8680125077565511
Seen States : 906
amirhossein@amirhossein-u:~/Desktop/CA1$ python3 Search.py < in4.txt
Path : [28, 19, 13, 3, 11, 24, 9, 23, 28, 23, 5, 7, 29]
Cost Of Path : 12
Execution Time : 32.02950962384542
Seen States : 3471
amirhossein@amirhossein-u:~/Desktop/CA1$ python3 Search.py < in5.txt
Path : [40, 42, 38, 24, 31, 45, 30, 48, 41, 18, 1, 19, 43, 49, 47, 49, 9, 34, 25, 50, 12, 16]
Cost Of Path : 21
Execution Time : 16.849187056223553
Seen States : 3478
amirhossein@amirhossein-u:~/Desktop/CA1$ python3 Search.py < in6.txt
Path : [1, 2, 3, 18, 19, 18, 3, 4, 3, 2, 1, 17, 16]
Cost Of Path : 16
Execution Time : 0.07090115547180176
Seen States : 178
amirhossein@amirhossein-u:~/Desktop/CA1$ python3 Search.py < in7.txt
Path : [1, 2, 1, 3, 1, 7, 1, 4]
Cost Of Path : 7
Execution Time : 0.008755922317504883
Seen States : 37
amirhossein@amirhossein-u:~/Desktop/CA1$
```

:IDS

این الگوریتم نیز uniformed است و نسبت به BFS کند تر است ولی جواب بهینه را ارائه می دهد و همچنین با توجه به این که برای پیمایش در هر عمق از DFS استفاده می کند به حافظه کمتری نسبت به BFS نیاز دارد. در هر دو الگوریتم BFS و IDS پیچیدگی زمان و حافظه O(b^d) است. با توجه به این که برای اجرای این الگوریتم با تست کیس های 4 و 5 به زمان خیلی زیادی نیاز بود، از آن ها صرف نظر کردیم.



```
amirhossein@amirhossein-u:~/Desktop/CA1$ python3 Search.py < in1.txt
Path : [1, 3, 4, 5, 7, 10, 11, 9, 8]
Cost Of Path : 8
Execution Time : 0.07791423797607422
Seen States : 267
amirhossein@amirhossein-u:~/Desktop/CA1$ python3 Search.py < in2.txt
Path : [9, 10, 2, 4, 12, 3, 7, 5, 8]
Cost Of Path : 8
Execution Time : 0.1244211196899414
Seen States : 358
amirhossein@amirhossein-u:~/Desktop/CA1$ python3 Search.py < in3.txt
Path : [13, 11, 10, 3, 2, 6, 12, 5, 9, 4, 1, 13, 11, 10]
Cost Of Path : 13
Execution Time : 11.485060850779215
Seen States : 6750
amirhossein@amirhossein-u:~/Desktop/CA1$ python3 Search.py < in6.txt
Path : [1, 2, 3, 4, 3, 18, 19, 18, 3, 2, 1, 17, 16]
Cost Of Path : 16
Execution Time : 2.1517271995544434
Seen States : 3295
amirhossein@amirhossein-u:~/Desktop/CA1$ python3 Search.py < in7.txt
Path : [1, 3, 6, 4, 1, 2, 1, 7]
Cost Of Path : 7
Execution Time : 0.0606255531311035 16
Seen States : 231
amirhossein@amirhossein-u:~/Desktop/CA1$
```

A^*:

بر خلاف دو الگوریتم قبل، این الگوریتم informed است و در هر مرحله راسی گسترش پیدا می کند که هزینه کمتری دارد. در استفاده از این الگوریتم نیاز به تعریف تابعی به نام $heuristic$ است که هزینه رسیدن به راس هدف را پیش بینی می کند. در واقع $h(n)$ پیش بینی هزینه رسیدن به راس هدف از راس n است. همچنین $g(n)$ نیز هزینه لازم برای رسیدن به راس n است.

حال $f(n) = h(n) + g(n)$ در نظر می گیریم و هر بار راسی با کمترین میزان f گسترش می یابد. برای این کار از یک heap استفاده می کنیم.

با توجه به تعریف $h(n)$ این الگوریتم می تواند جواب بهینه را ارائه بدهد یا ندهد. در واقع اگر $h(n)$ گونه ای تعریف شود که consistent باشد، جواب بهینه ارائه میشود. Consistent بودن تابع h در صورتی اثبات می شود که میزان هزینه پیش بینی شده بین دو راس توسط این تابع کمتر مساوی هزینه واقعی بین دو راس باشد.

اگر تابع h به خوبی تعریف شود و consistent باشد، این الگوریتم سریع تر از الگوریتم های قبل کار می کند در غیر این صورت ممکن است جواب بهینه را ارائه ندهد و یا زمان یافتن جواب نزدیک به الگوریتم BFS باشد.

تابع $heuristic$ را اینطور تعریف می کنیم که مقدار h(n) مجموع موارد زیر باشد :

- ابتدا نزدیک ترین دستور پختی را که به مریدش تحویل نداده شده را پیدا می کنیم و فاصله آن را به کمک BFS روی خود گراف بدست می آوریم و همچنین فاصله تمام دستور پخت های دیگر را که به مریدشان تحویل داده نشده است و سید حتما باید به آن رئوس برود به همین روش محاسبه می شود .
- به ازای هر یک دستور پختی که به مریدش تحویل داده نشده یک واحد محاسبه میشود.
- اگر در راس صعب العبور بودیم میزانی که باید تفکر کنیم تا بتوانیم از آن راس عبور کنیم محاسبه میشود.

کمترین هزینه برای رسیدن به استیت هدف این است که کمترین هزینه رسیدن به دستور پخت ها و رساندن به مریدشان را محاسبه کنیم، بنابراین در میان راه حتما باید به دستور پخت ها برسیم و سپس آن ها را به مریدشان تحویل دهیم که در این میان ممکن است از رئوس صعب العبور نیز بگذریم. بنابراین میزانی که در ابتدا به دست می آوریم بخشی از هزینه واقعی است و مقدار رساندن هر دستور پخت به مرید محاسبه نمی شود. بدین صورت واضح است که میزان هزینه پیش بینی شده توسط تابع کمتر مساوی میزان هزینه واقعی است و تابع h ، consistent می باشد.

```
amirhossein@amirhossein-u:~/Desktop/CA1$ python3 Search.py < in1.txt
Path : [1, 3, 4, 5, 7, 10, 11, 9, 8]
Cost Of Path : 8
Execution Time : 0.0097044308980306
Seen States : 30
amirhossein@amirhossein-u:~/Desktop/CA1$ python3 Search.py < in2.txt
Path : [9, 10, 6, 4, 12, 3, 7, 5, 8]
Cost Of Path : 8
Execution Time : 0.008818864822387695
Seen States : 26
amirhossein@amirhossein-u:~/Desktop/CA1$ python3 Search.py < in3.txt
Path : [13, 11, 10, 3, 2, 6, 12, 5, 9, 4, 1, 13, 11, 10]
Cost Of Path : 13
Execution Time : 0.11318437258402507
Seen States : 183
amirhossein@amirhossein-u:~/Desktop/CA1$ python3 Search.py < in4.txt
Path : [28, 19, 13, 3, 11, 24, 9, 2, 5, 7, 29, 22, 28]
Cost Of Path : 12
Execution Time : 8.687759955724081
Seen States : 1244
amirhossein@amirhossein-u:~/Desktop/CA1$ python3 Search.py < in5.txt
Path : [40, 42, 38, 24, 31, 45, 30, 48, 41, 18, 1, 19, 43, 49, 47, 49, 9, 34, 25, 50, 12, 16]
Cost Of Path : 21
Execution Time : 16.664238055547077
Seen States : 2899
amirhossein@amirhossein-u:~/Desktop/CA1$ python3 Search.py < in6.txt
Path : [1, 2, 3, 4, 3, 18, 19, 18, 3, 2, 1, 17, 16]
Cost Of Path : 16
Execution Time : 0.04056811332702637
Seen States : 104
amirhossein@amirhossein-u:~/Desktop/CA1$ python3 Search.py < in7.txt
Path : [1, 3, 1, 2, 1, 7, 1, 4]
Cost Of Path : 7
Execution Time : 0.008446455001831055
Seen States : 33
amirhossein@amirhossein-u:~/Desktop/CA1$
```

: Weighted A*

تفاوت در این الگوریتم این است که میزان h(n) در یک $\alpha > 0$ ضرب می شود، به عبارت دیگر :

$$f(n) = h(n) * \alpha + g(n)$$

این الگوریتم می تواند بسیار سریع تر از الگوریتم های قبل کار کند ولی ممکن است جواب بهینه نباشد و پاسخی نزدیک به پاسخ بهینه را در زمان کمتری به دست آورد:

$$\alpha = 1.6$$

```
amirhossein@amirhossein-u:~/Desktop/CA1$ python3 Search.py < in1.txt
Path : [1, 3, 4, 5, 7, 10, 11, 9, 8]
Cost Of Path : 8
Execution Time : 0.0060494740804036455
Seen States : 20
amirhossein@amirhossein-u:~/Desktop/CA1$ python3 Search.py < in2.txt
Path : [9, 10, 9, 4, 12, 3, 7, 5, 8]
Cost Of Path : 8
Execution Time : 0.003820180892944336
Seen States : 12
amirhossein@amirhossein-u:~/Desktop/CA1$ python3 Search.py < in3.txt
Path : [13, 11, 10, 3, 2, 6, 12, 5, 9, 4, 1, 13, 11, 10]
Cost Of Path : 13
Execution Time : 0.00702516237894694
Seen States : 20
amirhossein@amirhossein-u:~/Desktop/CA1$ python3 Search.py < in4.txt
Path : [28, 19, 3, 11, 24, 9, 23, 5, 7, 29, 20, 13, 23, 28]
Cost Of Path : 13
Execution Time : 0.04039446512858073
Seen States : 21
amirhossein@amirhossein-u:~/Desktop/CA1$ python3 Search.py < in5.txt
Path : [40, 42, 38, 24, 31, 45, 30, 48, 41, 18, 1, 19, 43, 49, 47, 49, 9, 34, 25, 50, 12, 16]
Cost Of Path : 21
Execution Time : 7.929087479909261
Seen States : 1875
amirhossein@amirhossein-u:~/Desktop/CA1$ python3 Search.py < in6.txt
Path : [1, 2, 3, 18, 19, 18, 3, 4, 3, 2, 1, 17, 16]
Cost Of Path : 16
Execution Time : 0.023828903834025066
Seen States : 59
amirhossein@amirhossein-u:~/Desktop/CA1$ python3 Search.py < in7.txt
Path : [1, 3, 1, 2, 1, 7, 1, 4]
Cost Of Path : 7
Execution Time : 0.00511940320332845
Seen States : 17
amirhossein@amirhossein-u:~/Desktop/CA1$
```

$$\alpha = 7$$

```
amirhossein@amirhossein-u:~/Desktop/CA1$ python3 Search.py < in1.txt
Path : [1, 3, 4, 5, 7, 10, 11, 9, 8]
Cost Of Path : 8
Execution Time : 0.0034429232279459634
Seen States : 12
amirhossein@amirhossein-u:~/Desktop/CA1$ python3 Search.py < in2.txt
Path : [9, 10, 9, 4, 12, 3, 7, 5, 8]
Cost Of Path : 8
Execution Time : 0.003771622975667318
Seen States : 12
amirhossein@amirhossein-u:~/Desktop/CA1$ python3 Search.py < in3.txt
Path : [13, 11, 10, 3, 2, 6, 12, 5, 10, 5, 9, 4, 1, 13, 11]
Cost Of Path : 14
Execution Time : 0.006589810053507487
Seen States : 19
amirhossein@amirhossein-u:~/Desktop/CA1$ python3 Search.py < in4.txt
Path : [28, 19, 3, 11, 24, 9, 23, 5, 7, 29, 20, 13, 19, 28]
Cost Of Path : 13
Execution Time : 0.02845422426859538
Seen States : 15
amirhossein@amirhossein-u:~/Desktop/CA1$ python3 Search.py < in5.txt
Path : [40, 42, 38, 24, 31, 45, 30, 48, 41, 18, 1, 19, 43, 49, 9, 49, 47, 49, 9, 34, 25, 50, 12, 16]
Cost Of Path : 23
Execution Time : 0.08907707532246907
Seen States : 83
amirhossein@amirhossein-u:~/Desktop/CA1$ python3 Search.py < in6.txt
Path : [1, 2, 3, 18, 19, 18, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]
Cost Of Path : 21
Execution Time : 0.010193109512329102
Seen States : 29
amirhossein@amirhossein-u:~/Desktop/CA1$ python3 Search.py < in7.txt
Path : [1, 3, 1, 2, 1, 7, 1, 4]
Cost Of Path : 7
Execution Time : 0.0025441646575927734
Seen States : 9
amirhossein@amirhossein-u:~/Desktop/CA1$
```

جمع بندی :

in1.txt

	پاسخ	میانگین زمان اجرا	تعداد استیت های دیده شده	مسیر
BFS	8	0.0086	35	[1, 3, 4, 5, 7, 10, 11, 9, 8]
IDS	8	0.077	267	[1, 3, 4, 5, 7, 10, 11, 9, 8]
A*	8	0.0097	30	[1, 3, 4, 5, 7, 10, 11, 9, 8]
A* (α=1.6)	8	0.00604	20	[1, 3, 4, 5, 7, 10, 11, 9, 8]
A* (α=7)	8	0.00344	12	[1, 3, 4, 5, 7, 10, 11, 9, 8]

in2.txt

	پاسخ	میانگین زمان اجرا	تعداد استیت های دیده شده	مسیر
BFS	8	0. 0.029	90	: [9, 10, 9, 4, 12, 3, 7, 5, 8]
IDS	8	0.124	358	[9, 10, 2, 4, 12, 3, 7, 5, 8]
A*	8	0.0088	26	[9, 10, 6, 4, 12, 3, 7, 5, 8]
A* (α=1.6)	8	0.00382	12	[9, 10, 9, 4, 12, 3, 7, 5, 8]
A* (α=7)	8	0.00377	12	[9, 10, 9, 4, 12, 3, 7, 5, 8]

in3.txt

	پاسخ	میانگین زمان اجرا	تعداد استیت های دیده شده	مسیر
BFS	13	0.868	906	[13, 11, 10, 3, 2, 6, 12, 5, 9, 4, 1, 13, 11, 10]
IDS	13	11.485	6750	[13, 11, 10, 3, 2, 6, 12, 5, 9, 4, 1, 13, 11, 10]
A*	13	0.1131	183	[13, 11, 10, 3, 2, 6, 12, 5, 9, 4, 1, 13, 11, 10]
A* (α=1.6)	13	0.00702	20	[13, 11, 10, 3, 2, 6, 12, 5, 9, 4, 1, 13, 11, 10]
A* (α=7)	14	0.00658	19	[13, 11, 10, 3, 2, 6, 12, 5, 10, 5, 9, 4, 1, 13, 11]

in4.txt

	پاسخ	میانگین زمان اجرا	تعداد استیت های دیده شده	مسیر
BFS	12	32.02	3471	[28, 19, 13, 3, 11, 24, 9, 23, 28, 23, 5, 7, 29]
IDS	----	----	----	----
A*	12	8.687	1244	[28, 19, 13, 3, 11, 24, 9, 2, 5, 7, 29, 22, 28]
A* (α=1.6)	13	0.0403	21	[28, 19, 3, 11, 24, 9, 23, 5, 7, 29, 20, 13, 23, 28]
A* (α=7)	13	0.0284	15	[28, 19, 3, 11, 24, 9, 23, 5, 7, 29, 20, 13, 19, 28]

in5.txt

	پاسخ	میانگین زمان اجرا	تعداد استیت های دیده شده	مسیر
BFS	21	16.84	3478	[40, 42, 38, 24, 31, 45, 30, 48, 41, 18, 1, 19, 43, 49, 47, 49, 9, 34, 25, 50, 12, 16]
IDS	----	----	----	----
A*	21	16.664	2899	[40, 42, 38, 24, 31, 45, 30, 48, 41, 18, 1, 19, 43, 49, 47, 49, 9, 34, 25, 50, 12, 16]
A* (α=1.6)	21	7.929	1875	[40, 42, 38, 24, 31, 45, 30, 48, 41, 18, 1, 19, 43, 49, 47, 49, 9, 34, 25, 50, 12, 16]
A* (α=7)	23	0.089	83	[40, 42, 38, 24, 31, 45, 30, 48, 41, 18, 1, 19, 43, 49, 9, 49, 47, 49, 9, 34, 25, 50, 12, 16]

in6.txt

	پاسخ	میانگین زمان اجرا	تعداد استیت های دیده شده	مسیر
BFS	16	0.07	178	[1, 2, 3, 18, 19, 18, 3, 4, 3, 2, 1, 17, 16]
IDS	16	2.151	3295	[1, 2, 3, 4, 3, 18, 19, 18, 3, 2, 1, 17, 16]
A*	16	0.0405	104	[1, 2, 3, 4, 3, 18, 19, 18, 3, 2, 1, 17, 16]
A* (α=1.6)	16	0.02382	59	[1, 2, 3, 18, 19, 18, 3, 4, 3, 2, 1, 17, 16]
A* (α=7)	21	0.0101	29	[1, 2, 3, 18, 19, 18, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]

in7.txt

	پاسخ	میانگین زمان اجرا	تعداد استیت های دیده شده	مسیر
BFS	7	0.0087	37	[1, 2, 1, 3, 1, 7, 1, 4]
IDS	7	0.06	231	[1, 3, 6, 4, 1, 2, 1, 7]
A*	7	0.0084	33	[1, 3, 1, 2, 1, 7, 1, 4]
A* (α=1.6)	7	0.00511	17	[1, 3, 1, 2, 1, 7, 1, 4]
A* (α=7)	7	0.0025	9	[1, 3, 1, 2, 1, 7, 1, 4]

** در حل برخی از مشکلات از سایت های https://stackoverflow.com و https://www.geeksforgeeks.org استفاده شده است.