پروژهٔ پایانی درس طراحی الگوریتم‌ها

نگار موقتیان، ۹۸۳۱۰۶۲

۲.

مساله ۸ وزیر با مانع:

حل این مساله با استفاده از روش‌هایی که برای حل بهینهٔ مسائل بررسی شد (مانند روش تقسیم و غلبه، برنامه نویسی پویا و ...) قابل انجام نیست، بنابراین به جستجوی فضای حالات مساله با روش backtracking می‌پردازیم. در مسالهٔ ۸ وزیر بدون مانع دو وزیر نمی‌توانند در سطر و یا ستونی یکسان قرار بگیرند و می‌توانیم از این طریق فضای حالات مساله را محدود کرده و الگوریتم را بهبود دهیم. اما در این مساله به دلیل وجود موانع ممکن است بتوانیم بیش از یک وزیر در هر سطر یا ستون داشته باشیم (در حالات خاص حتی ممکن است بتوانیم $n/2$ وزیر در یک سطر یا ستون داشته باشیم) بنابراین محدود سازی فضای حالات از روش قبلی امکان پذیر نیست و ناچار به بررسی تمامی حالات هستیم.

برای حل این مساله از آرایه‌ای با اندازهٔ n (به نام p) برای مشخص کردن محل وزیرها استفاده شده‌است. این آرایه مانند یک شمارنده n رقمی عمل می کند که هر رقم آن می‌تواند از 0 تا n^2 تغییر یابد. البته برای بهینه سازی این الگوریتم و در نظر نگرفتن حالات تکراری فرض می‌شود مکان وزیر اول پیش از وزیر دوم، و وزیر دوم پیش از وزیر سوم و به همین ترتیب است. یعنی همواره:

$$0 \le p[0] < p[1] < \ldots < p[n] < n^2$$

از وزیر اول شروع می‌کنیم. این وزیر را در اولین خانهٔ ممکن قرار می‌دهیم ($p[0] = 0$). سپس وزیر دوم را در خانهٔ پس از آن قرار می‌دهیم و بررسی می‌کنیم آیا توسط وزیر اول تهدید می‌شود یا خیر. اگر تهدید می‌شد آن را به خانهٔ بعدی می‌بریم ($p[1]$ را یک عدد افزایش می‌دهیم) و این کار را تا زمانی ادامه می‌دهیم که وزیر دوم توسط وزیر اول تهدید نشود. سپس وزیر سوم را در خانهٔ بعد از وزیر دوم قرار می‌دهیم و بررسی می‌کنیم که آیا توسط وزیر اول و یا دوم تهدید می‌شود یا خیر و در صورتی که تهدید می‌شد آن را به خانهٔ بعد می‌بریم و به همین ترتیب ادامه می‌دهیم.

هنگام انجام این عملیات اگر وزیر i ام از خانه‌های شطرنج خارج شد به این معناست که حالتی برای این وزیر وجود نداشته که توسط وزیرهای قبلی تهدید نشود، لذا مکان این وزیر را صفر می‌کنیم و به سراغ وزیر قبلی (وزیر i-1) می‌رویم و آن را یک خانه به جلو می‌بریم (backtracking). این عملیات مانند قبل باید تا جایی ادامه پیدا کند که وزیر i-1 ام دیگر تهدید نشود. پس از مشخص شدن جای این وزیر مانند قبل الگوریتم را ادامه می‌دهیم و به وزیر i ام برمی‌گردیم.

اگر توانستیم وزیر آخر را در جایی قرار دهیم که توسط وزیرهای قبلی‌اش تهدید نشود یک پاسخ برای مساله پیدا کرده‌ایم و آن را چاپ می‌کنیم. پس از چاپ کردن این پاسخ باز وزیر آخر را یک خانه به جلو می‌بریم و مانند قبل ادامه می‌دهیم تا پاسخ‌های احتمالی دیگر را بیابیم.

در نهایت هنگامی که تمامی حالات ممکن را برای وزیر اول بررسی کرده بودیم یعنی تمام فضای حالت بررسی شده‌است و می‌توانیم برنامه را خاتمه دهیم.

در ادامه توابع استفاده شده در این برنامه به صورت دقیق‌تر بررسی شده‌اند:

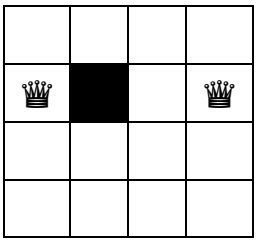
تابع displayAnswer:

تنها وظیفهٔ این تابع چاپ کردن پاسخی است که با قرار دادن آخرین وزیر می‌یابیم. به ازای هر خانهٔ خالی یک مربع توخالی، به ازای هر وزیر یک علامت وزیر و به ازای هر مانع یک مربع توپر چاپ می‌شود.

تابع isThreatened:

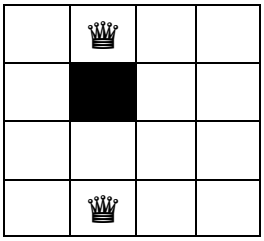
با گرفتن دو وزیر i و k بررسی می‌کند که این دو وزیر یکدیگر را تهدید می‌کنند یا خیر. دو وزیر در سه حالت ممکن است یکدیگر را تهدید کنند:

۱. دو وزیر در یک ردیف قرار داشته باشند؛ کافیست شماره ردیف دو وزیر را با یکدیگر مقایسه کنیم. در مرحلهٔ بعد اگر این شرط برقرار بود بررسی می‌کنیم آیا مانعی بین این دو قرار دارد که این تهدید را رفع کند یا خیر. این مانع نیز باید در همان ردیف قرار داشته باشد و شماره ستون آن مابین شماره ستون دو وزیر باشد. اگر مانعی با این شرایط پیدا شد خطر رفع می‌شود، در غیر این صورت دو وزیر یکدیگر را تهدید می‌کنند و این تابع true برمی‌گرداند.



۲. دو وزیر در یک ستون قرار داشته باشند؛ کافیست شماره ستون دو وزیر را با یکدیگر مقایسه کنیم. در مرحلهٔ بعد اگر این شرط برقرار بود بررسی می‌کنیم آیا مانعی بین این دو قرار دارد که این تهدید را رفع کند یا خیر. این مانع نیز باید در همان ستون قرار داشته باشد و شماره ردیف آن مابین شماره ردیف دو وزیر باشد. اگر مانعی با این شرایط پیدا شد خطر رفع می‌شود، در غیر این صورت دو وزیر

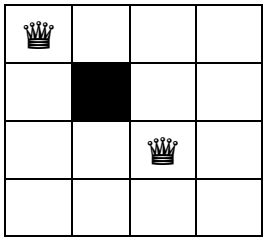
یکدیگر را تهدید می‌کنند و این تابع true برمی‌گرداند.



۳. دو وزیر بر روی یک قطر قرار داشته باشند؛ در این حالت وزیر i و k باید شرط زیر را داشته باشند:

$$|x_i - x_k| = |y_i - y_k|$$

در مرحلهٔ بعد اگر این شرط برقرار بود بررسی می‌کنیم آیا مانعی بین این دو قرار دارد که این تهدید را رفع کند یا خیر. این مانع نیز باید شرط بالا را با مختصات وزیرها داشته باشد و شماره ردیف و همچنین شماره ستون آن مابین شماره ردیف و ستون دو وزیر باشد. اگر مانعی با این شرایط پیدا شد خطر رفع می‌شود، در غیر این صورت دو وزیر یکدیگر را تهدید می‌کنند و این تابع true برمی‌گرداند.



در نهایت اگر هیچ یک از شروط بالا خطری برای دو وزیر ایجاد نمی‌کرد تابع false برمی‌گرداند.

تابع backtrack:

این تابع بررسی می‌کند که آیا تمام حالات برای هر وزیر بررسی شده‌است یا خیر. در صورتی که تمامی حالات بررسی شده بود مکان این وزیر را صفر می‌کنیم و وزیر قبلی را به جلو حرکت می‌دهیم. این کار را به قدری تکرار می‌کنیم که حالات جدیدی برای بررسی کردن وجود داشته باشد و پس از آن الگوریتم را ادامه می‌دهیم. همچنین طی این عملیات اگر متوجه شدیم که وزیر اول نیز تمامی حالات خود را طی کرده بدین معناست که تمامی فضای حالات بررسی شده و باید الگوریتم را خاتمه دهیم.

تحلیل زمانی الگوریتم:

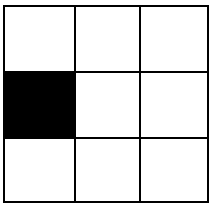
از آن‌جایی که این الگوریتم تمامی فضای حالات را بررسی می‌کند زمان اجرای نمایی دارد زیرا برای هر وزیر n^2 حالت قابل تصور است (هر چند با توجه به شرطی که بیان شد در عمل تعداد کمتری از حالات بررسی می‌شوند). علاوه بر آن برای بررسی اینکه هر وزیر دیگری را تهدید می‌کند یا خیر نیاز به پیمایش تمامی وزیرهای پیشین و موانع داریم. بنابراین زمان دقیق‌تر اجرای این الگوریتم برابر است با:

$$(n^2-m)(0.m)\times(n^2-m-1)(1.m)\times\ldots\times(n^2-m-(n-1))((n-1).m)=\prod_{i=0}^{n-1}(n^2-m-i)(i.m)$$
$$\leq((n^2-m)(n.m))^n\Rightarrow T(n)\in O(mn^{3n})$$

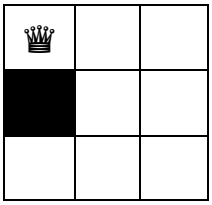
که در آن n ابعاد صفحهٔ شطرنج و m تعداد موانع است.

مثالی از این الگوریتم:

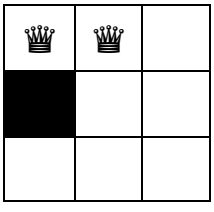
فرض کنید یک صفحه 3x3 داریم که یک مانع مانند زیر در آن قرار گرفته‌است:



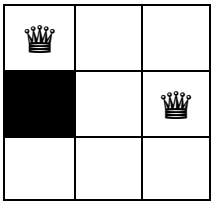
طبق الگوریتم گفته شده ابتدا وزیر اول را در خانهٔ اول قرار می‌دهیم.



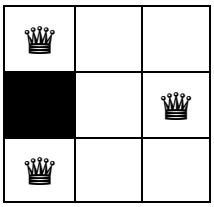
در این‌جا وزیر اول تنها وزیر است و وزیر دیگری آن را تهدید نمی‌کند پس وزیر دوم را در خانهٔ پس از آن قرار می‌دهیم:



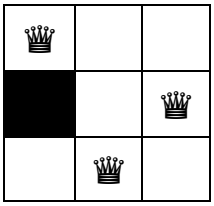
این بار طبق حالت اول از حالات گفته شده دو وزیر یکدیگر را تهدید می‌کنند پس وزیر را باز هم به جلو حرکت می‌دهیم (در این جا ۴ بار این حرکت را تکرار می‌کنیم) تا به خانه‌ای برسیم که وزیر دوم توسط وزیر اول تهدید نشود.



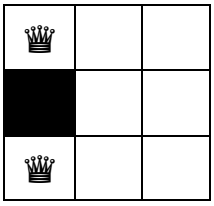
حال می‌توانیم وزیر سوم را نیز در خانهٔ بعد از وزیر دوم قرار دهیم.



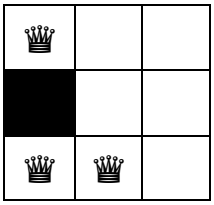
هیچ یک از دو وزیر قبل وزیر سوم را تهدید نمی‌کنند و در مجموع نیاز به قراردهی سه وزیر داشتم، پس در این مرحله یکی از پاسخ‌های مساله را پیدا کرده‌ایم و آن را چاپ می‌کنیم. اما ممکن است این مساله پاسخ‌های دیگری نیز داشته باشد، پس وزیر آخر را بار دیگر به جلو حرکت می‌دهیم و الگوریتم را ادامه می‌دهیم تا به پاسخ‌های احتمالی دیگر برسیم.



در این مکان وزیر سوم توسط وزیر دوم طبق حالت سوم تهدید می‌شود پس باید باز هم وزیر سوم را به جلو حرکت دهیم. در خانهٔ بعد نیز وزیر سوم توسط وزیر دوم طبق حالت دوم تهدید می‌شود و بار دیگر آن را به جلو حرکت می‌دهیم. اما این بار وزیر سوم از جدول خارج می‌شود و این به این معناست که تمام حالات آن بررسی شده‌اند و حالت مطلوبی پیدا نشده، پس یک مرحله به عقب برمی‌گردیم و وزیر دوم را به جلو می‌بریم.

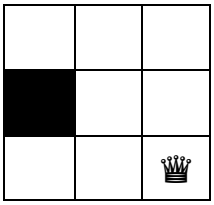


در این حالت وزیر اول وزیر دوم را تهدید نمی‌کند پس می‌توانیم وزیر سوم را در خانهٔ پس از آن قرار دهیم.



اما باز هم وزیر سوم توسط وزیر دوم تهدید می‌شود و باید آن را به جلو حرکت دهیم.

باز هم طی چند مرحله وزیر سوم از صفحه خارج می‌شود و باید وزیر دوم را به جلو حرکت دهیم. به همین ترتیب تمامی حالات وزیرهای دوم و سوم بررسی می‌شوند و پس از آن باید وزیر اول را به جلو حرکت دهیم. پس از آن مراحل الگوریتم با تغییر مکان وزیر اول تکرار می‌شوند و طی چندین مرحله به حالت زیر می‌رسیم:



وزیر اول در خانهٔ آخر قرار می‌گیرد و جایی برای قرار دهی وزیرهای دیگر نمی‌ماند. در این مرحله می‌توان الگوریتم را خاتمه داد و ادعا کرد مساله پاسخ دیگری ندارد.

نمونهٔ ورودی و خروجی برنامه:

شکل روبه‌رو نمونهٔ ورودی و خروجی برنامه مطابق با مثال بالا می‌باشد:

```
Size of the board: 3
Number of obstacles: 1
Position of the obstacle number 1: 2 1
-------------------------------------

♛ □ □
■ □ ♛
♛ □ □

Total number of answers: 1

Process finished with exit code 0
```