فصل ۷

روش‌های جست‌وجو

در بخش‌های قبل روش‌های تقسیم‌وحل، پویا، و حریصانه که عمدتاً برای به دست آوردن راه‌حل‌های سریع و چندجمله‌ای برای مسایل استفاده می‌شوند، مورد بررسی قرار گرفت. ولی پیدا کردن چنین الگوریتم‌هایی کار ساده‌ای نیست و همه‌ی مسایل را نمی‌توان با روش‌های مذکور حل کرد.

اگر مسئله‌ای با هیچ یک از روش‌های فوق، قابل حل نبود، ممکن است تنها راه‌حل آن جست‌وجوی فضای حالت باشد؛ یعنی برای پیدا کردن جواب، کلیه‌ی حالت‌های مسئله را مورد بررسی قرار می‌دهیم. این جست‌وجو معمولا به یکی از روش‌های زیر انجام می‌گیرد:

۱.۷ روش پس‌گرد

پس‌گرد[1] روشی است که کلیه‌ی فضای حالت قابل قبول را با نظم خاصی ایجاد و جست‌وجو می‌کند. به این صورت که در مرحله‌هایی که با چندین انتخاب روبرو می‌شود، یکی از انتخاب‌ها را با فرض درست بودن دنبال می‌کند، اگر به جواب نرسید، با پس‌گرد (backtrack) انتخاب خود را عوض می‌کند و این کار را تا تمام شدن انتخاب‌ها ادامه می‌دهد. این روش می‌تواند با یافتن اولین جواب متوقف شود ویا این که جست‌وجو را برای یافتن کلیه‌ی جواب‌ها ادامه دهد.

الگوریتم‌های پس‌گرد معمولا دارای هزینه‌ی نمایی می‌باشند.

---

[1] backtracking

۱.۱.۷ مسئله‌ی هشت‌وزیر

```
EIGHTQUEENS()
for i1 ← 1 to 8
    do for i2 ← 1 to 8
            do for i3 ← 1 to 8
                    do ...
                        for i8 ← 1 to 8
                            do try ← (i1, i2, ..., i8)
                                if SOLUTION (try)
                                  then PRINT try
```

بردار k-promising

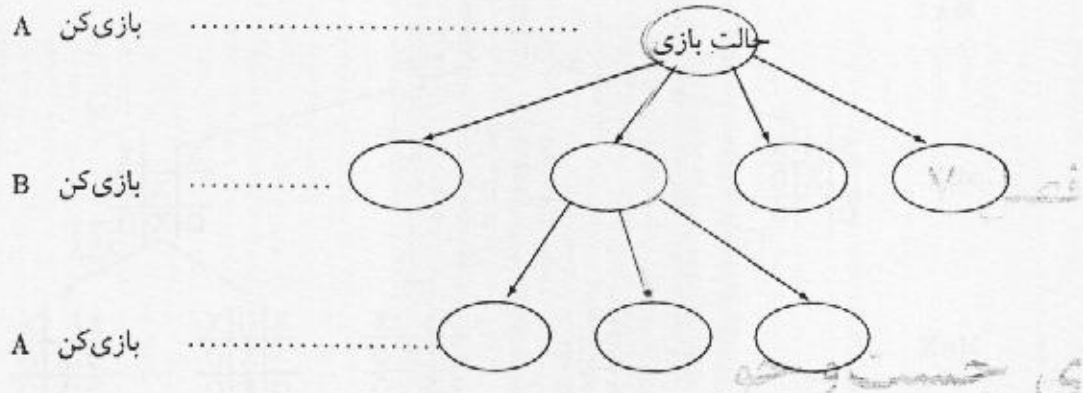
```
N-QUEENS(k, col, diag45, diag135)
    ▷ try is k-promising array [1..k]
    ▷ col ← {try[i] : 1 ≤ i ≤ k}
    ▷ diag45 ← {try[i] − i + 1 : 1 ≤ i ≤ k}
    ▷ diag135 ← {try[i] + i − 1 : 1 ≤ i ≤ k}
if k = N {an N-promising verstor is a solution}
  then PRINT try
  else { find (k + 1)-promising extensioun}
       for j ← 1 to N
           do if j ∉ col and j − k ∉ diag45 and j + k ∉ diag135
                then try[k + 1] ← j
                N-QUEENS(k + 1, col + {j}, diag45 + {j − k}, diag135 + {j + k})
```

برای $n = ۱۲$ ۴۷۹۰۰۱۶۰۰ جای‌گشت وجود دارد. اولین جواب در ۴۵۴۶۰۴۴امین حلقه به‌دست می‌آید. ولی درخت حالت در روش پس‌گرد ۸۵۶۱۸۹ گره دارد که در ۲۶۲امین گره یک جواب به‌دست می‌آید.

۲.۷ درخت بازی

یکی از روش‌های جست‌وجوی فضای مسئله، استفاده از درخت بازی[۲] می‌باشد که جهت تعیین استراتژی برد و انجام به‌ترین بازی ممکن در هر مرحله به‌کار می‌رود. هر گره درخت بر یک وضعیت از بازی دلالت می‌کند و وضعیت‌هایی که با حرکت بعد قابل تولید هستند، به عنوان فرزندان این گره در نظر گرفته می‌شوند (مانند شکل ۱.۷).

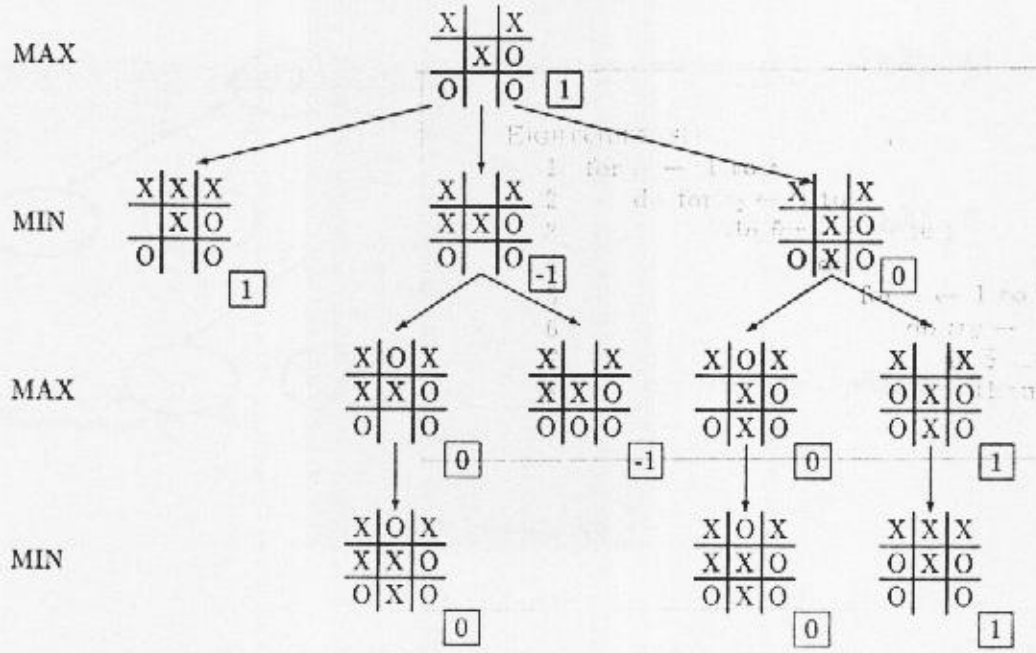
[۲] game tree



شکل ۱.۷: درخت بازی.

برگ‌ها خاتمه‌ی بازی را نشان می‌دهند و هر مسیر تا رسیدن به یک برگ، بیان‌گر یک بازی است که یک یال این مسیر توسط بازیکن اول انتخاب شده و یال بعد را بازیکن دوم انتخاب کرده است. به هر گره عددی نسبت داده می‌شود که بستگی به وضعیت بازی دارد. برای روشن شدن مطلب به مثال زیر توجه کنید:

در بازی X-O از درخت بازی می‌توان به صورت نشان داده شده در شکل ۲.۷ در تصمیم‌گیری برای رسیدن به هدف نهایی که قرار دادن حروف مشابه به صورت دنبال هم می‌باشد، استفاده نمود. در حالت نهایی (برگ‌ها) اگر X برنده باشد عدد ۱، اگر O (حریف) برنده باشد عدد ۱-، و در غیر این صورت عدد صفر به آن برگ نسبت داده می‌شود. در سطح پدر اگر حرکت (انتخاب یال) از X باشد بیشینه‌ی عدد فرزندان به این گره نسبت داده می‌شود و اگر حرکت از O باشد، کمینه‌ی آنها. بدین معنی که ما وضعیتی را انتخاب می‌کنیم که بیشترین امتیاز را دارد و حریف هم به‌ترین بازی خود را ارایه می‌دهد تا کم‌ترین امتیاز نصیب ما شود. از این رو این درخت را min-max game tree نیز می‌گویند.

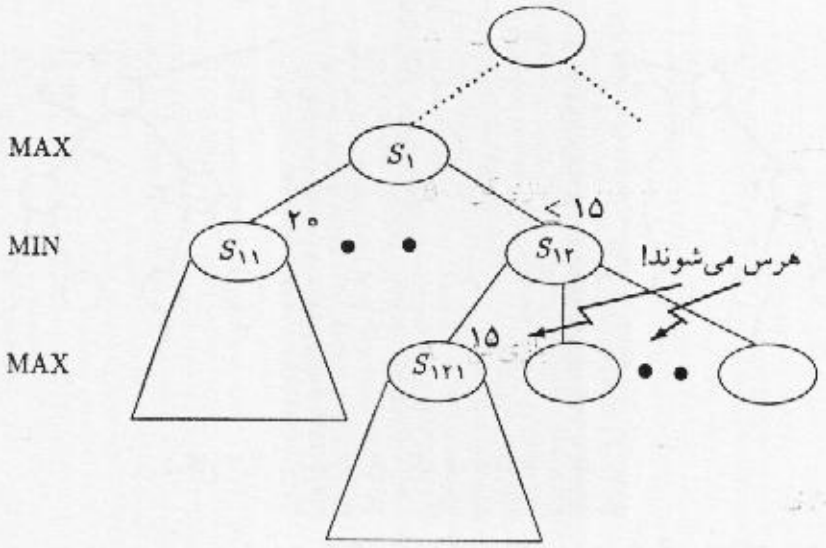


شکل ۲.۷: درخت بازی در بازی X-O

۱.۲.۷ پیاده‌سازی درخت بازی

```
SEARCH(B, mode)
    ▷ B is is a board and mode is either max or min
    ▷ It returns a real value for maximum score
1   if B is a leaf
2      then return PAYOFF(B)
3      else if mode = Max
4              then value ← −1
5              else value ← +1
6           for each child C of board B
7               do if mode = max
8                     then value ← max(value, SEARCH(C, min))
9                     else value ← min(value, SEARCH(C, max)) return value
```

مسئله‌ی فوق تمام فضای حالت را جست‌وجو می‌کند، اما در بعضی موارد به دلیل زیاد بودن حالت‌ها چنین امری امکان پذیر نیست. مثلا در بازی شطرنج ساخت درخت تا رسیدن به حالت‌های نهایی که بتوان به آن امتیاز نسبت داد مقدور نیست، لذا هر چند سطح که امکان دارد، ساخته می‌شود و در سطح آخر با توجه به وضعیت



شکل ۳.۷: با بازکردن S_{121} بی‌تاثیر بودن S_{21} مشخص می‌گردد و بقیه‌ی فرزندان آن هرس می‌شوند.

مهره‌ها در صفحه، عددی را حدس زده، وبه آن نسبت می‌دهیم. هرقدر که این حدس قوی‌تر و تعداد سطوح ایجاد شده‌ی درخت بیش‌تر باشد، به همین میزان بازی به‌تری انجام می‌گیرد.

۳.۷ محدود کردن فضای جست‌وجو

در مسایل فوق سعی بر این بود که تمامی فضای حالت مورد جست‌وجو قرار گیرد. اما با استفاده از روش‌های زیر می‌توان فضای جست‌وجو را کاهش داد:

۱. هرس کردن[۳]

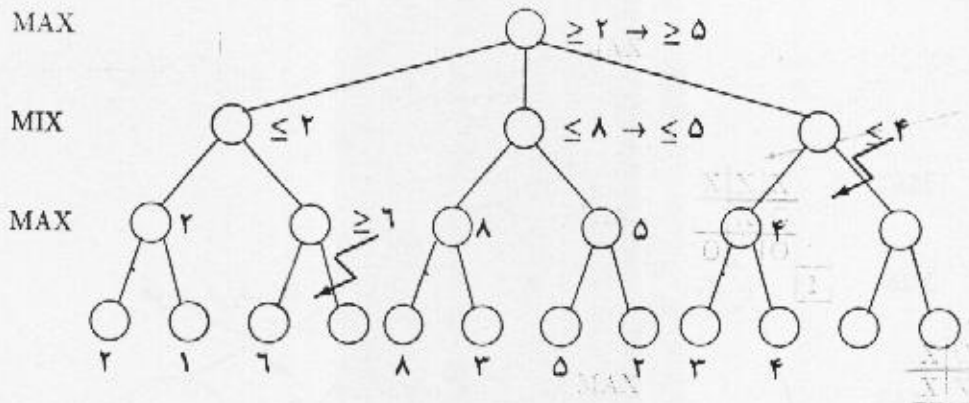
۲. انشعاب و تحدید[۴]

۱.۳.۷ هرس کردن

گاهی اوقات جست‌وجوی برخی از شاخه‌ها بی‌تاثیر است و می‌توان از جست‌وجوی آن صرف نظر کرد. در شکل ۳.۷ چون S_{11} امتیازی معادل ۲۰ دارد، درنتیجه امتیاز S1 بیشتر از ۲۰ خواهد بود و در شاخه‌ی S_{12} پس از باز کردن S_{121} با امتیاز ۱۵ متوجه می‌شویم که امتیاز S_{12} کمتر از ۱۵ خواهد بود، و از آن‌جا که امتیاز S_1 بیش‌تر از ۲۰ است لذا از باز کردن بقیه‌ی فرزندان S_{121} صرف نظر می‌کنیم و آن‌ها را با تمامی زیرشاخه‌هایشان هرس می‌کنیم.

---

[۳] pruning
[۴] Branch & Bound



شکل ۴.۷: نمونه‌ای از یک α-β pruning

در مورد درخت بازی این روش را «هرس α-β»[۵] گویند. شکل ۴.۷ نمونه‌ای از آن را نشان می‌دهد.

۲.۳.۷ انشعاب و تحدید

در این روش نیز مانند هرس کردن، از بازکردن برخی از شاخه‌ها خودداری می‌شود. به این صورت که با تجزیه و تحلیل اطلاعات محلی موجود در بعضی از مرحله‌ها، مشخص می‌گردد که از این شاخه نتیجه‌ی مطلوبی حاصل نخواهد شد، لذا از باز کردن آن صرف نظر می‌شود. هم‌چنین با توجه به همین اطلاعات، تصمیم می‌گیریم که ابتدا کدام یک از شاخه‌های درخت را مورد جست‌وجو قرار دهیم. برای روشن شدن مطلب به مثال زیر توجه کنید:

۳.۳.۷ فروشنده‌ی دوره‌گرد

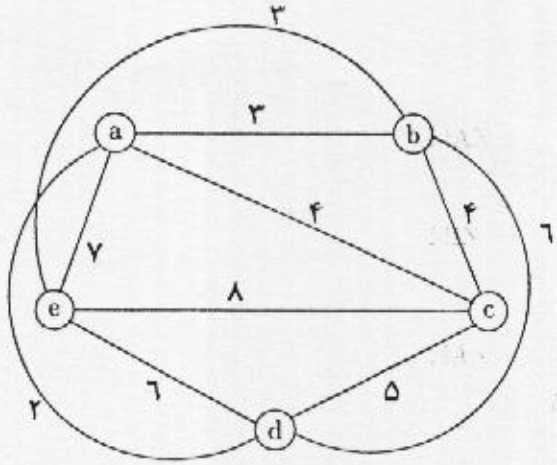
فرض کنید شهرها و راه‌های بین آن‌ها مطابق شکل ۵.۷ داده شده است. هدف پیدا کردن دوری با حداقل وزن ممکن است، که از کلیه‌ی رأس‌ها عبور کند و از هر راس فقط یک بار بگذرد.

حالت مسئله: تعدادی از یال‌ها انتخاب شده‌اند، پس قطعاً این یال‌ها در مسیر سفر وجود دارند. هم‌چنین تعدادی از یال‌ها حذف شده‌اند، که نمی‌توان از آنها در مسیر سفر استفاده کرد. مثلاً کلیه‌ی مسیرهایی که یال‌های ab و cd را شامل می‌شود و یال ac در آن مسیرها وجود ندارد، یک حالت از مسئله می‌باشد. این حالت را به صورت زیر نمایش می‌دهیم:

$$ab, cd, \overline{ac}$$

تعیین حد: در هر حالت برای هر رأس دو یال با کمترین وزن، از «یال‌های موجود» انتخاب می‌کنیم. بدیهی‌است که هیچ مسیر هامیلتونی پیدا نمی‌شود که وزن آن کم‌تر از نصف مجموع وزن‌های این یال‌ها باشد.

[۵] α-β pruning



شکل ۵.۷: مسیر شهرها در مسئله‌ی فروشنده‌ی دوره‌گرد.

این حد همان اطلاعات محلی هر حالت محسوب می‌شود. مثلاً در حالتی که هیچ یالی حذف نشده است، حد به طریق زیر محاسبه می‌شود:

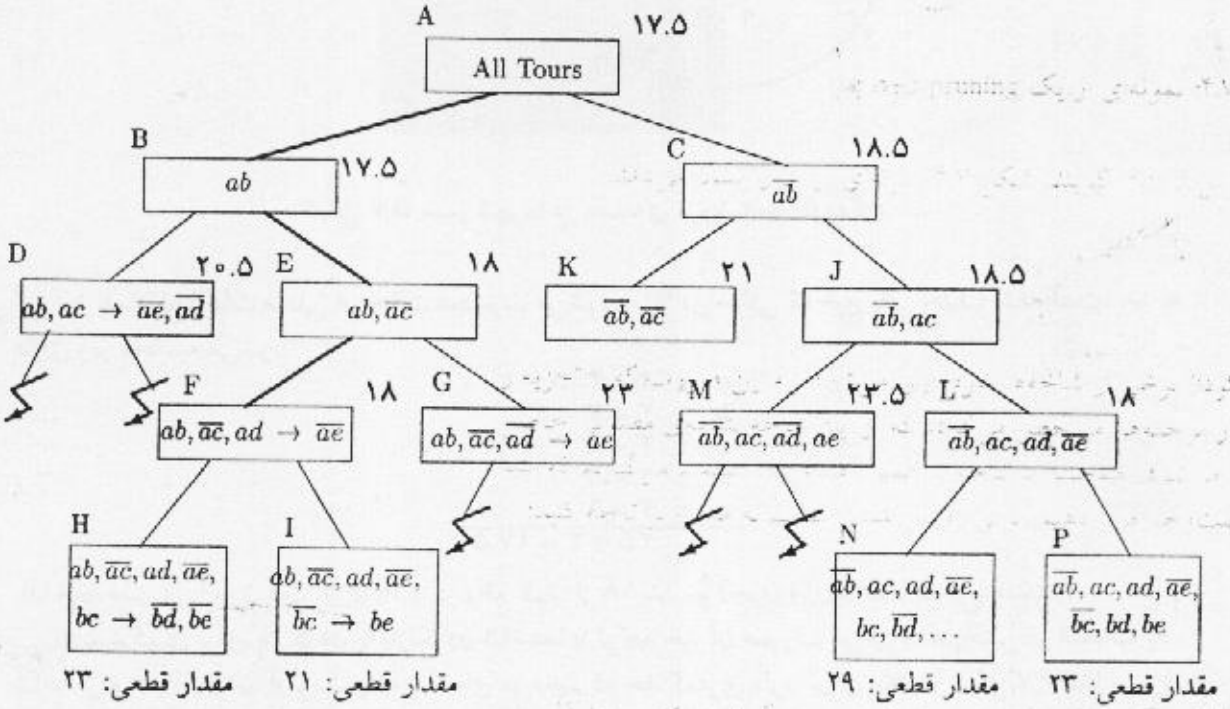
$$\begin{array}{lcl} a & : & ۲+۳ \\ b & : & ۳+۳ \\ c & : & ۴+۴ \\ d & : & ۲+۵ \\ e & : & ۳+۶ \\ \hline & & ۳۵ \div ۲ = ۱۷.۵ \end{array}$$

لذا هیچ مسیری با وزن کمتر از ۱۷/۵ و در واقع کم‌تر از ۱۸ نداریم (چون وزن‌ها عدد صحیح هستند.)

تصمیم‌گیری برای باز کردن یا بازنکردن شاخه‌ها با توجه حد آن صورت می‌گیرد. هم‌چنین در انتخاب یک شاخه برای دنبال کردن، اولویت را به شاخه‌ای می‌دهیم که حد کمتری دارد. برای مثال در شکل ۶.۷ ابتدا مسیری که پررنگ‌تر رسم شده، جست‌وجو می‌شود.

در شکل فوق پس از رسیدن به I با ارزش ۲۱ و بازگشت به مرحله‌ی E، مرحله G هرس می‌شود. زیرا مسیرهایی که از G تولید می‌شوند، بزرگتر یا مساوی ۲۳ هستند. و هم‌چنین در مرحله‌ی D مسیرهایی که تولید می‌شوند، بزرگ‌تر یا مساوی ۲۱ هستند لذا D هم هرس می‌شود. در یک سطح بالاتر چون ۲۱ از ۱۹ بزرگتر است، لذا اعمال فوق را دوباره تکرار می‌کنیم و اتفاقاً با توجه به شکل مسیری به طول ۱۹ بدست می‌آید که جواب مسئله است در حالی که D,G,M,K باز نشده‌اند.

نکته‌ی مهم در حل این گونه مسایل به دست آوردن یک حد (Bound) صحیح می‌باشد.



شکل ۶.۷: درخت جست‌وجوی مسئله‌ی فروشنده‌ی دوره‌گرد.