به نام خدا

طراحی الگوریتم‌ها

آرش شفیعی

جستجوی ترکیباتی

جستجوی ترکیبیاتی

- روش‌های تقسیم و حل، برنامه‌ریزی پویا و حریصانه را برای حل دسته‌ای از مسئله‌های محاسباتی بررسی کردیم. در مسئله‌های بررسی شده، یک راه‌حل ساده برای پیدا کردن جواب مسئله جستجوی همهٔ حالت‌ها است ولی چنین جستجویی در زمان معقول (زمان چند جمله‌ای) امکان پذیر نیست چرا که تعداد همهٔ حالت‌های ممکن از مرتبه نمایی است و بنابراین جستجوی همهٔ حالت‌ها در زمان نمایی صورت می‌گیرد. توسط روش‌های تقسیم و حل، برنامه‌ریزی پویا و حریصانه روش‌هایی برای حل مسئله‌ها در زمان معقول ارائه کردیم.
- پیدا کردن جواب در زمان چند جمله‌ای برای همهٔ مسئله‌ها همیشه امکان پذیر نیست و گاهی تنها راه حل ، جستجوی فضای حالت برای آن مسئله است. جستجوی فضای حالت به معنی تولید همهٔ جواب‌های مسئله و انتخاب حالت بهینه یا حالت مورد نظر از بین همهٔ حالت‌های ممکن است. تعداد این حالت‌ها از مرتبهٔ نمایی است و بنابراین چنین الگوریتم‌هایی به زمان نمایی برای محاسبه نیاز دارند و در نتیجه برای مسائلی با اندازهٔ بزرگ در عمل قابل استفاده نیستند.

جستجوی ترکیبیاتی

- در مبحث جستجوی ترکیبیاتی [1] ، روش‌های جستجوی همه فضای حالت برای یک مسئله بررسی می‌شود. در این روش‌ها مطالعه می‌کنیم چگونه به طور منظم همهٔ حالت‌ها را بررسی کنیم و یا اینکه چگونه با استفاده از روش‌هایی فضای حالت را محدود کنیم.

- ترکیبیات [2] شاخه‌ای از ریاضیات است که در آن به بررسی ساختارهای متناهی و شمارش این ساختارها می‌پردازد.

- برای مثال گراف یک ساختار متناهی است و تعداد مسیرها در یک گراف، متناهی است. برای پیدا کردن بلندترین مسیر در یک گراف می‌توانیم همهٔ مسیرها را بررسی و بلندترین آنها را انتخاب کنیم.

- بهینه سازی ترکیبیاتی [3] شاخه‌ای از بهینه‌سازی است که در آن مجموعهٔ جواب‌های امکان پذیر گسسته است و هدف پیدا کردن جواب بهینه از بین همهٔ جواب‌های ممکن است.

---

[1] combinatorial search
[2] combinatorics
[3] combinatorial optimization

جستجوی ترکیبیاتی

- روش پسگرد [1] روشی است که با استفاده از آن همهٔ فضای حالت را جستجو می‌کنیم. در این روش با یکی از حالت‌ها در فضای حالت شروع می‌کنیم و برای انتخاب حالت بعد یکی از پارامترهای فضای حالت را تغییر می‌دهیم. ممکن است در انتخاب حالت بعد، چند پارامتر قابل تغییر باشند. در اینصورت یکی از پارامترها را تغییر می‌دهیم و به حالت بعد می‌رویم و سپس پسگرد می‌کنیم تا یکی دیگر از پارامترها را تغییر دهیم. بدین صورت همهٔ حالت‌ها در فضای حالت بررسی می‌شوند.

[1] backtrack

جستجوی ترکیبیاتی

- برای مثال فرض کنید می‌خواهیم همهٔ رشته‌ها با طول ۲ که از سه حرف a و b و c تشکیل شده‌اند را بشماریم. یا به عبارت دیگر همهٔ فضای حالت را تولید کنیم.
- با استفاده از روش پسگرد در ابتدا ۳ انتخاب برای حرف اول داریم حرف a را انتخاب می‌کنیم و سپس از بین ۳ انتخاب برای حرف دوم، حرف a را انتخاب می‌کنیم پس رشتهٔ aa را به دست می‌آوریم، سپس پسگرد می‌کنیم و حرف b را برای حرف دوم رشته انتخاب می‌کنیم و رشته ab را بدست می‌آوریم. بار دیگر با یک پسگرد رشتهٔ ac را بدست می‌آوریم. در پسگرد بعدی هیچ انتخابی برای حرف دوم وجود نخواهد داشت پس دوباره پسگرد می‌کنیم و حرف b را به عنوان حرف اول انتخاب می‌کنیم. با استفاده از این روش به ترتیب رشته‌های ba ، bb ، bc ، ca ، cb و cc به دست می‌آیند.

پسگرد

- فرض کنید وارد یک باغ پر پیچ و خم [1] شده‌اید و می‌خواهید راه خروجی را پیدا کنید. راه را در پیش می‌گیرید و به هر چند راهی که می‌رسید راه اول از سمت چپ را انتخاب می‌کنید. در پایان یا راه خروجی را پیدا می‌کنید و یا به بن‌بست بر می‌خورید. در صورتی که به بن‌بست رسیدید، بازمی‌گردید تا به اولین چندراهی قبل از بن‌بست برسید. به جای راه اول از سمت چپ، دومین راه از سمت چپ را امتحان می‌کنید و در صورت برخورد با بن‌بست راه را باز می‌گردید و در چند راهی راه سوم را امتحان می‌کنید. فرض کنید که همهٔ راه‌ها در چند راهی آخر را امتحان کردید و به بن‌بست خوردید. در این صورت باید مسیر را بازگردید تا به دومین چند راهی قبل از بن‌بست برسید و این بار در دومین چندراهی قبل از بن‌بست، دومین راه را انتخاب کنید. این روند را ادامه می‌دهید تا راه خروجی را پیدا کنید. به این روش حل مسئله روش پسگرد گفته می‌شود.

[1] maze

پسگرد

- روش پسگرد وقتی استفاده می‌شود که می‌خواهیم مسئله‌ای را حل کنیم که در آن عناصر یک دنباله [1] باید از اشیائی از یک مجموعهٔ [2] معین انتخاب شوند، به طوری‌که دنباله ویژگی مشخصی داشته باشد و معیار [3] معینی را برآورده کند.

[1] sequence
[2] set
[3] criterion

مسئلهٔ چند وزیر

- می‌خواهیم تعداد n وزیر را در یک صفحه شطرنج $n \times n$ به گونه‌ای قرار دهیم که هیچ یک از وزیرها همدیگر را تهدید نکنند. به عبارت دیگر هیچ دو وزیری نباید در یک سطر یا ستون یا خط مورب مشترکی قرار بگیرند.

- دنباله‌ای که در این مسئله به دنبال آن می‌گردیم، دنباله‌ای است از n مکان که n وزیر در آنها قرار گرفته‌اند. مجموعه‌ای که هر یک از عناصر دنباله می‌توانند از اعضای آن مجموعه انتخاب شوند عبارت است از مجموعه‌ای شامل n^2 مکان بر روی صفحهٔ شطرنج.

- ویژگی معینی که این دنباله باید داشته باشد این است که هیچ دو مکان انتخاب شده‌ای بر روی یک خط افقی، عمودی یا مورب قرار نگیرد.

- مسئلهٔ چند وزیر یا n وزیر، حالت کلی مسئلهٔ ۸ وزیر است که در آن ۸ وزیر در یک صفحه شطرنج استاندارد با تعداد ۸ × ۸ مکان قرار می‌گیرند.

- در اینجا برای سهولت نمایش حالات مختلف مسئله ۴ وزیر را در نظر می‌گیریم.

مسئلهٔ چند وزیر

- روش پسگرد در واقع روشی است که در یک درخت [1] به صورت عمق–اول [2] جستجو می‌کند. پس ابتدا جستجوی عمق–اول در درخت را توضیح می‌دهیم.
- فرض کنید درختی داریم که از تعدادی رأس و یال تشکیل شده است و می‌خواهیم همهٔ مسیرهای درخت را که با ریشه شروع می‌شوند و با یک برگ پایان می‌یابند بررسی کنیم.
- در واقع دنباله‌هایی که در اینجا بررسی می‌کنیم دنباله‌هایی هستند که یک مسیر در درخت را نشان می‌دهند و با ریشه آغاز و با یکی از برگ‌ها پایان می‌یابند.

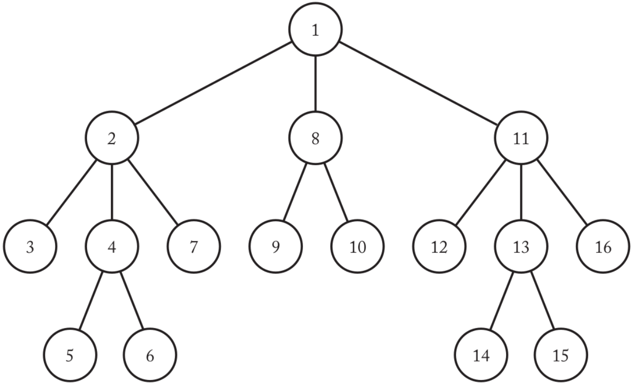
---
[1] tree
[2] depth-first

مسئلهٔ چند وزیر

- ابتدا به سراغ ریشهٔ درخت می‌رویم و ریشه را به عنوان اولین عنصر دنباله انتخاب می‌کنیم. سپس اولین فرزند از سمت چپ در سطح یک درخت را انتخاب می‌کنیم و اگر این رأس دارای فرزند بود اولین فرزند آن را در سطح دو انتخاب می‌کنیم. این کار را ادامه می‌دهیم تا به یک برگ برسیم. تا اینجا یک مسیر در درخت را بررسی کرده‌ایم. با فرض اینکه برگ در سطح n قرار دارد، به رأس پدر در سطح $n-1$ باز می‌گردیم و فرزند دوم را انتخاب می‌کنیم. این کار را ادامه می‌دهیم تا مسیر دوم و به همین ترتیب همهٔ مسیرها در درخت را بررسی کنیم.

مسئلهٔ چند وزیر

- درخت زیر را در نظر بگیرید. با استفاده از روش پسگرد ابتدا مسیر $(1, 2, 3)$ سپس $(1, 2, 4, 5)$ سپس $(1, 2, 4, 6)$ ، و به همین ترتیب $(1, 2, 7)$ ، $(1, 8, 9)$ ، $(1, 8, 10)$ ، الی آخر بررسی می‌شوند.



مسئلهٔ چند وزیر

- الگوریتم زیر این روش پسگرد را نشان می‌دهد.

Algorithm Depth First TreeSearch

```
function DEPTH-FIRST-TREE-SEARCH(node v)
visit(v);
for each child u of v do
    Depth-First-Tree-Search(u)
```

مسئلهٔ چند وزیر

- حال که با روش پسگرد آشنا شدیم، مسئلهٔ ۴ وزیر را در نظر می‌گیریم. می‌خواهیم ۴ وزیر را در یک صفحهٔ شطرنج ۴×۴ به گونه‌ای قرار دهیم که هیچ دو وزیری یکدیگر را تهدید نکنند. از آنجایی که هیچ دو وزیری نمی‌توانند در یک سطر قرار بگیرند، پس هر وزیر را باید در یک سطر متفاوت از بقیه وزیرها قرار دهیم. از آنجایی که هر وزیر در هر یک از ستون‌ها می تواند قرار بگیرد، بنابراین تعداد همهٔ حالت‌هایی که باید بررسی شوند برابر است با $۴ \times ۴ \times ۴ \times ۴ = ۲۵۶$.

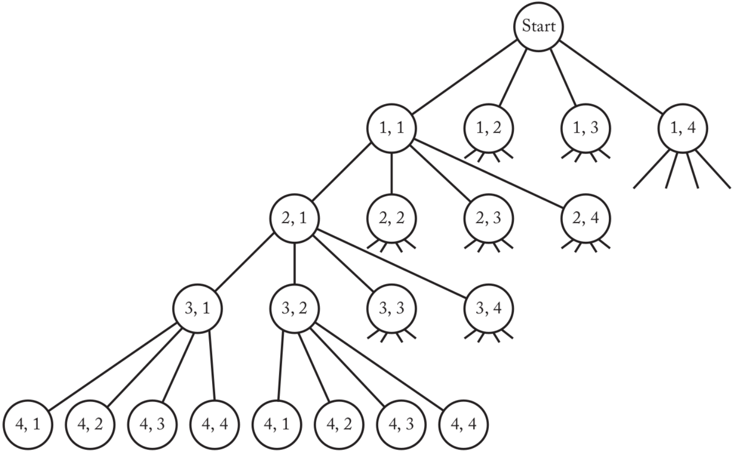
- برای بررسی همهٔ حالت‌ها درختی تشکیل می‌دهیم که در آن، مکان اولین وزیر در سطح یک انتخاب می‌شود و مکان وزیر دوم در سطح دوم و به همین ترتیب مکان وزیر سوم در سطح سوم و مکان وزیر چهارم در سطح چهارم انتخاب می‌شوند.

- یک مسیر از ریشه تا برگ درواقع یکی از گزینه‌های انتخاب مکان‌های صفحه را نشان می‌دهد. یک گزینه جواب مسئله است اگر در آن هیچ دو وزیری یکدیگر را تهدید نکنند.

- به درخت تشکیل شده درخت فضای حالت گفته می‌شود.

مسئلهٔ چند وزیر

- قسمتی از درخت حالت مسئله ۴ وزیر در شکل زیر نمایش داده شده است.



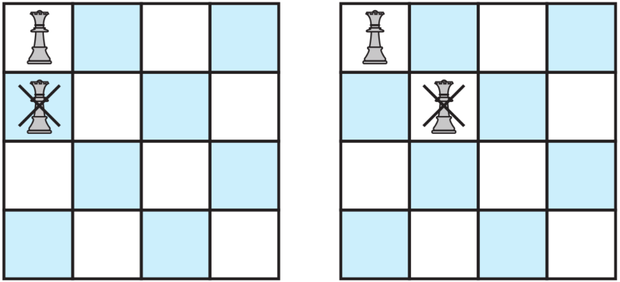
مسئلهٔ چند وزیر

- این درخت در مجموع ۲۵۶ برگ دارد که هر مسیر از ریشه تا یکی از برگ‌ها یکی از گزینه‌ها را نشان می‌دهد. در هریک از رأس‌های درخت یک جفت (i, j) ذخیره می‌شود که برابر با یک مکان در صفحهٔ شطرنج در سطر i و ستون j است.

مسئلهٔ چند وزیر

- جستجو در این درخت می تواند بهینه‌تر از بررسی همهٔ ۲۵۶ انتخاب نیز انجام شود. برای مثال وقتی وزیر اول در مکان $(1,1)$ قرارگرفت، بدیهی است که وزیر دوم نمی‌تواند در مکان $(2,1)$ قرار بگیرد پس این مسیر در درخت ادامه داده نمی‌شود. همینطور وزیر دوم نمی‌تواند در مکان دوم قرار بگیرد پس نیازی نداریم این مسیر را نیز ادامه دهیم.

- در شکل زیر این بهینه‌سازی نشان داده شده است.



مسئلهٔ چند وزیر

- پسگرد عبارت است از روندی که توسط آن وقتی در یک انتخاب به یک گزینهٔ غیر جواب می‌رسیم، در درخت فضای حالت به عقب بازمی‌گردیم یا به عبارت دیگر از انتخاب یک رأس صرف نظر می‌کنیم و به رأس پدر پسگرد می‌کنیم تا فرزند بعدی پدر را انتخاب کنیم.
- به یک رأس در درخت فضای حالت نومید کننده [1] می‌گوییم، اگر انتخاب آن رأس به جواب نیانجامد و به همین ترتیب به یک رأس امید دهنده [2] می‌گوییم اگر با انتخاب آن همچنان احتمال رسیدن به جواب وجود داشته باشد.

[1] nonpromising
[2] promising

مسئلهٔ چند وزیر

- به طور خلاصه در روش پسگرد، بر روی درخت فضای حالت، جستجوی عمق اول انجام می‌دهیم و در فرایند جستجو اگر به رأس نومید کننده برخورد کردیم مسیر را ادامه نمی‌دهیم و به رأس پدر پسگرد می‌کنیم.
- به این روش هرس‌کردن [1] فضای حالت نیز گفته می‌شود که در آن تعدادی از دنباله‌ها بررسی نمی‌شوند.

[1] pruning

مسئلهٔ چند وزیر

- در حالت کلی این الگوریتم به صورت زیر نوشته می‌شود.

Algorithm Checknode

```
function CHECKNODE(node v)
if promising(v) then
   if there is a solution at v then
      write the solution
   else
      for each child u of v do
         Checknode(u)
```

مسئلهٔ چند وزیر

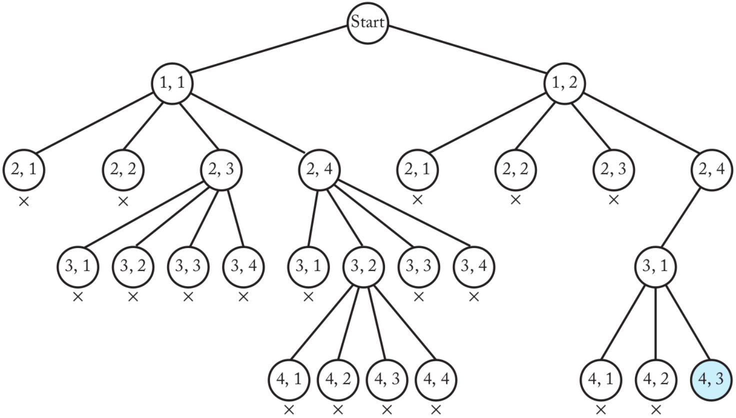
- ریشهٔ فضای حالت به تابع Checknode داده می‌شود که توسط آن رأس ریشه بررسی می‌شود. در بررسی یک رأس، ابتدا باید بررسی شود که انتخاب آن رأس امید دهنده است یا نومید کننده. اگر انتخاب آن امید دهنده بود و به جواب رسیدیم، جواب را چاپ می‌کنیم. اگر انتخاب امید دهنده بود ولی به جواب نرسیدیم، رئوس فرزند به ترتیب برای بررسی انتخاب می‌شود.

مسئلهٔ چند وزیر

- تابع promising برای مسئله‌های مختلف متفاوت است. در مسئلهٔ ۴ وزیر، این تابع مقدار نادرست باز می‌گرداند اگر مکان‌های انتخاب شده از ریشه تا رأس مورد بررسی، به صورت سطری، ستونی یا قطری در یک راستا باشند.

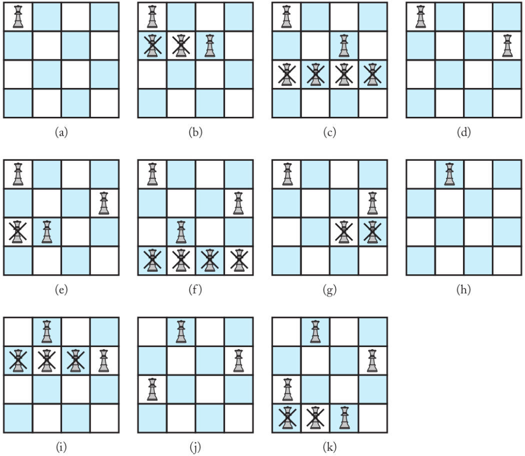
مسئلهٔ چند وزیر

- در شکل زیر قسمتی از درخت فضای حالت به صورت هرس شده نشان‌داده شده است.



مسئلهٔ چند وزیر

- در شکل زیر روند بررسی فضای حالت در صفحهٔ شطرنج نشان‌داده شده است.



مسئلهٔ چند وزیر

- تابع promising باید بررسی کند که آیا دو وزیر در یک ستون یا قطر قرارگرفته‌اند یا خیر.
- اگر $\mathrm{col}(i)$ ستونی باشد که وزیر i ام در آن قرارگرفته است، باید بررسی کنیم که $\mathrm{col}(i)$ و $\mathrm{col}(j)$ برای هیچ دو وزیر i و j برابر نباشد.
- همچنین دو وزیر به صورت مورب یکدیگر را تهدید می‌کنند اگر $\mathrm{col}(i) - \mathrm{col}(j) = i - j$ و یا $\mathrm{col}(i) - \mathrm{col}(j) = j - i$ باشد.

مسئلهٔ چند وزیر

- الگوریتم پسگرد برای مسئلهٔ چند وزیر به صورت زیر است. برای شروع الگوریتم تابع `Queens(0)` فراخوانی می‌شود.

Algorithm Queens

```
function QUEENS(index i)
index j
if promising(i) then
    if i == n then
        print col[1] through col[n]
    else
     ▷ See if queen in (i+1)-st row can be
     ▷ positioned in each of the n columns.
        for j = 1 ; j <= n ; j++ do
            col[i + 1] = j
            queens(i + 1)
```

مسئلهٔ چند وزیر

Algorithm Promising

```
function PROMISING(index i)
index k = 1
boolean prom = true
  ▷ Check if any queen k threatens queen in the i-th row.
while k < i and prom do
    if col[i] == col[k] or |col[i] - col[k]| == i - k then
        prom = false
    k++
return prom
```

مسئلهٔ چند وزیر

- حال می‌خواهیم الگوریتم پسگرد برای چند وزیر را تحلیل کنیم. برای این کار تعداد رئوسی که در درخت فضای حالت بررسی می‌شوند را محاسبه می‌کنیم. از آنجایی که به دست آوردن تعداد دقیق حالات وقتی حالت‌ها هرس می‌شوند ساده نیست، برای تعداد رئوس بررسی شده در درخت فضای حالت یک کران بالا در نظر می‌گیریم.

- در درخت فضای حالات در سطح صفر، یک رأس داریم، در سطح یک تعداد n رأس، در سطح ۲ تعداد n^2 رأس و در سطح n تعداد n^n رأس داریم. پس تعداد همهٔ رئوس بررسی شده برابر است با :

$$1 + n + n^2 + n^3 + \cdots + n^n = \frac{n^{n+1} - 1}{n - 1}$$

- برای مسئله ۸ وزیر این تعداد برابر است با

$$\frac{8^{8+1} - 1}{8 - 1} = 19,173,961$$

مسئلهٔ چند وزیر

- حال یک کران بالای دیگر برای تعداد رئوس بررسی شده در نظر می‌گیریم. می‌دانیم که هیچ دو وزیری نمی‌توانند در یک ستون قرار بگیرند، بنابراین وقتی وزیر اول انتخاب شد، وزیر دوم تنها در ۷ مکان می‌تواند قرار بگیرد پس در سطح اول ۸ رأس و در سطح دوم ۷ × ۸ رأس، در سطح سوم ۶ × ۷ × ۸ رأس داریم و بدین ترتیب الی آخر.
- پس در حالت کلی کران بالای رئوس بررسی شده برابراست با :

$$1 + n + n(n-1) + n(n-1)(n-2) + \cdots + n!$$

- برای مسئله ۸ وزیر این تعداد برابراست با ۱۰۹،۶۰۱ رأس.

مسئلهٔ چند وزیر

- تعداد دقیق رئوس بررسی شده را می‌توانیم با پیاده‌سازی الگوریتم به دست آوریم.
- از آنجایی که کران بالای تعداد حالت مورد بررسی $n!$ است، زمان اجرای الگوریتم پسگرد برای مسئلهٔ n وزیر $O(n!)$ است.

شاخه و کران

- روش شاخه و کران [1] برای بهبود الگوریتم‌های پسگرد به کار می‌روند.
- روش شاخه و کران، همچون روش پسگرد درخت فضای حالت را برای یافتن جواب بررسی می‌کند.
- یک الگوریتم شاخه و کران در هر رأس درخت جستجوی حالات، کرانی را محاسبه می‌کند که با استفاده از مقدار کران می‌توان گفت آیا آن رأس امید دهنده است یا خیر. مقداری که به عنوان کران در هر رأس محاسبه می‌شود، با استفاده از کرانی بر روی جواب مسئله به دست می‌آید.

[1] branch and bound

شاخه و کران

- این روش معمولاً برای مسائل بهینه‌سازی استفاده می‌شود. در مسائل بهینه‌سازی هدف یافتن جواب بهینه (کوچکترین، بزرگترین، ...) است. در هر لحظه در هنگام پیمایش درخت فضای حالت یکی از جواب‌های به دست آمده تا آن لحظه بهینه است. بنابراین قبل از بسط دادن یک رأس می‌توانیم محاسبه کنیم آیا جوابی که با بسط دادن آن رأس به دست می‌آید، از جواب بهینه‌ای که تا آن لحظه به دست آمده است، بهتر است یا خیر. در صورتی که امیدی به یافتن جواب بهتر نبود، رأس مورد پیمایش بسط داده نمی‌شود.

- بنابراین اگر با بسط دادن یک رأس امید به یافتن جواب بهینه‌تر وجود نداشت، می‌گوییم آن رأس نومیدکننده [1] است، در غیر اینصورت امیددهنده [2] است.

[1] nonpromising
[2] promising

شاخه و کران

- می‌توانیم مسئلهٔ کوله پشتی ۰-۱ را با استفاده از روش پسگرد حل کنیم، و در این صورت می‌توان از روش شاخه و کران استفاده کرد، بدین ترتیب که تابع promising (برای بررسی اینکه یک رأس امیددهنده است یا خیر) مقدار نادرست باز می‌گرداند اگر در یک رأس حداکثر ارزشی که می‌تواند از برداشتن باقی اشیاء حاصل شود، از مقدار بیشینهٔ ارزش به دست آمده تا آن لحظه بیشتر نباشد.

شاخه و کران

- در روش شاخه و کران، با استفاده از مقدار کران به دست آمده، نه تنها می‌توان تصمیم گرفت که یک رأس بسط داده شود و یا خیر، بلکه می‌توان علاوه بر آن با استفاده از کران به دست آمده، تصمیم گرفت کدام رأس برای بسط دادن مناسب‌تر است.
- با استفاده از این روش معمولاً می‌توان با سرعت بیشتری به جواب بهینه دست پیدا کرد.
- به این روش جستجوی بهتر اول [1] با هرس کردن شاخه و کران [2] گفته می‌شود.

[1] best–first search
[2] branch and bound pruning

شاخه و کران

- در جستجوی درخت فضای حالت گاهی به جای جستجوی عمق اول، از جستجوی سطح اول [1] استفاده می‌شود. در این روش ابتدا ریشه بررسی می‌شود، سپس رئوس سطح یک و پس از آن رئوس سطح دو و بدین ترتیب همهٔ رئوس تا برگ‌ها بررسی می‌شوند.
- برخلاف جستجوی عمق اول که در آن از یک الگوریتم بازگشتی استفاده می‌شود، در جستجوی سطح–اول از یک صف برای پیمایش رئوس درخت استفاده می‌کنیم.
- بدین ترتیب فرزندان یک رأس در صف قرار می‌گیرند و به ترتیب فرزندان یک به یک از صف خارج شده و فرزندانشان پیمایش می‌شوند. بنابراین در این روش ابتدا سطح اول درخت، سپس سطح دوم و به همین ترتیب همهٔ سطوح پیمایش می‌شوند. به همین دلیل به این جستجو سطح–اول گفته می‌شود.

[1] breadth–first search

شاخه و کران

- الگوریتم زیر الگوریتم جستجوی درخت با استفاده از روش سطح-اول را نشان می‌دهد.

Algorithm Breadth-First-Tree-Search

```
function BREADTH-FIRST-TREE-SEARCH(tree T)
queue Q
node u, v
initialize(Q)  ▷ initialize Q to be empty.
u = root(T)
visit(u)
enqueue(Q, u)
while !empty(Q) do
    v = dequeue(Q)
    for each child u of v do
        visit(u)
        enqueue(Q, u)
```

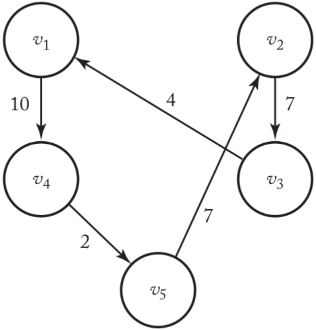
مسئلهٔ فروشندهٔ دوره‌گرد

- یک فروشندهٔ دوره‌گرد می‌خواهد برای فروش اجناس خود به همهٔ شهرها سفر کند. هر دو شهر می‌توانند توسط یک جادهٔ یک طرفه به یکدیگر متصل شده باشند و هر جاده طولی معین دارد. فروشندهٔ دوره‌گرد می‌خواهد از شهر خود سفر را آغاز کند و مسیری را بپیماید که از هر شهر تنها یک بار عبور کند و در پایان به شهر خود بازگردد. در ضمن می‌خواهد مسیر پیموده شده کوتاهترین مسیر باشد.
- این مسئله را با استفاده از یک گراف مدلسازی می‌کنیم. در یک گراف جهت‌دار، می‌خواهیم کوتاه‌ترین مسیری را پیدا کنیم که از یک رأس آغاز می‌شود، از هریک از رئوس تنها یک بار عبور می‌کند و به رأس اول باز می‌گردد. چنین مسیری یک مسیر بهینه است. از آنجایی که این مسیر بهینه از همهٔ رئوس عبور می‌کند، بنابراین می‌توانیم از هر رأسی مسیر را آغاز کنیم. به مسیری که از هر رأس تنها یک بار عبور می‌کند مسیر همیلتونی می‌گوییم و به مسیری که از هر رأس تنها یک بار عبور کند و به رأس اول بازگردد یک دور همیلتونی می‌گویند. در اینجا به دنبال کوتاهترین دور همیلتونی می‌گردیم.

مسئلهٔ فروشندهٔ دوره‌گرد

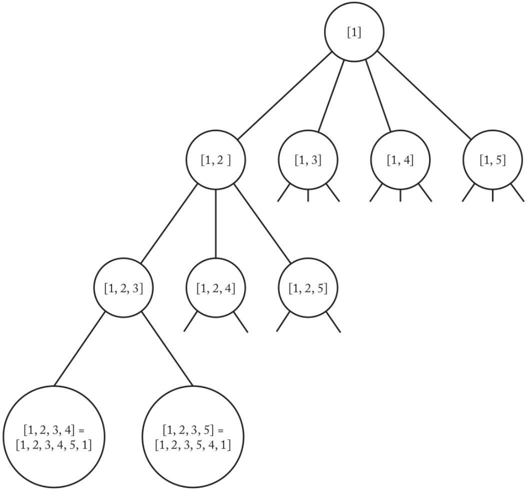
- در شکل زیر ماتریس مجاورت برای یک گراف جهت‌دار نشان داده شده است که در آن از هر رأس به رأس دیگر یک یال وجود دارد. اعداد در ماتریس مجاورت طول یال‌ها از یک رأس به رأس دیگرند. کوتاهترین دور همیلتونی در این گراف نشان داده شده است.

$$\begin{bmatrix} 0 & 14 & 4 & 10 & 20 \\ 14 & 0 & 7 & 8 & 7 \\ 4 & 5 & 0 & 7 & 16 \\ 11 & 7 & 9 & 0 & 2 \\ 18 & 7 & 17 & 4 & 0 \end{bmatrix}$$



مسئلهٔ فروشندهٔ دوره‌گرد

- قسمتی از درخت جستجوی فضای حالت برای این مسئله در زیر نشان داده شده است.



مسئلهٔ فروشندهٔ دوره‌گرد

- در این درخت فضای حالت، با رأس شماره ۱ ازگراف آغاز می‌کنیم. مسیر بهینه ممکن است از هر یک از رئوس ۲ ، ۳ ، ۴ و ۵ عبورکند، بنابراین به ازای هریک از این رئوس یک فرزند در سطح یک در درخت فضای حالت می‌سازیم. رأس $[1,2]$ در درخت فضای حالت مسیری را درگراف مشخص می‌کندکه از رأس ۱ و ۲ درگراف عبورکند.

مسئلهٔ فروشندهٔ دوره‌گرد

- اکنون باید برای هر رأس در درخت فضای حالت یک مقدار کران پیدا کنیم. در هر رأس درخت فضای حالت یک کران پایین برای طول مسیری که می‌توان با بسط دادن آن رأس به دست آورد محاسبه می‌کنیم.
- اگر کران پایین محاسبه شده در یک رأس از کوتاهترین مسیر همیلتونی به دست آمده تا آن لحظه کمتر باشد، آن رأس درخت فضای حالت امیددهنده است و آن رأس را بسط می‌دهیم، در غیر اینصورت آن رأس نومیدکننده است و بسط دادن را از آن رأس ادامه نمی‌دهیم.

مسئلهٔ فروشندهٔ دوره‌گرد

- برای محاسبهٔ کران به صورت زیر عمل می‌کنیم.

- در هر رأس درخت فضای حالت تعدادی از رئوس گراف پیمایش شده و تعدادی پیمایش نشده‌اند. به ازای رأس‌های پیمایش شده در گراف طول مسیر معین شده است. اما به ازای رأس‌های پیمایش نشده در گراف باید یک کران پایین برای طول مسیر محاسبه کنیم. رأس پیمایش نشدهٔ V_i را در نظر بگیرید. جهت پیدا کردن یک کران پایین برای کوتاهترین مسیر، به ازای هریک از رأس‌های پیمایش نشدهٔ V_i باید $\min_k(V_i, V_k)$ را محاسبه کنیم. به عبارت دیگر به ازای هر یک از رئوس پیمایش نشدهٔ V_i باید یالی را پیدا کنیم که از V_i خارج می‌شود و کمترین هزینه را دارد. مجموع طول این یال‌ها یک کران پایین برای طول مسیر است. توجه کنید که این بدین معنا نیست که یال با کمترین هزینه در مسیر همیلتونی انتخاب می‌شود، بلکه بدین معناست که هیچ مسیر همیلتونی با طول کمتر از مقدار محاسبه شده وجود نخواهد داشت.

مسئلهٔ فروشندهٔ دوره‌گرد

- محاسبه کران را با یک مثال بررسی می‌کنیم. در ریشه درخت فضای حالت هیچ‌یک از رئوس گراف پیموده نشده‌اند. کران پایین مسیر در ریشه را می‌توانیم توسط رابطهٔ $\sum_i \min_k(V_i, V_k)$ محاسبه کنیم. این مقدار برابراست با :

$$\sum_i \min_k(V_i, V_k) = 4 + 7 + 4 + 2 + 4 = 21$$

- بنابراین طول کوتاه‌ترین مسیر ممکن با شروع از رأس V_1 برابر است با ۲۱ .

- حال به ازای همهٔ فرزندان ریشه یک کران محاسبه می‌کنیم و رأسی را برای پیمایش انتخاب می‌کنیم که کران محاسبه شده برای آن کمینه باشد.

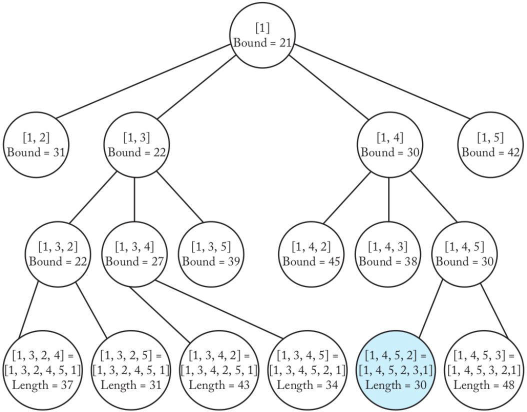
مسئلهٔ فروشندهٔ دوره‌گرد

- رأس $[1,2]$ در درخت فضای حالت را در نظر بگیرید. در این حالت یالی با طول 14 از رأس V_1 به رأس V_2 پیموده شده است.
- طول کوتاهترین مسیری که با شروع از این مسیر می‌تواند وجود داشته باشد برابر خواهد بود با ۱۴ به علاوه کوتاهترین مسیری که از بقیه رئوس به غیر از V_1 و V_2 عبور می‌کند.
- کران پایین مسیر در این رأس برابراست با :

$$14 + \sum_{i \in \{2,3,4,5\}} \min_k(V_i, V_k) = 14 + 7 + 4 + 2 + 4 = 31$$

مسئلهٔ فروشندهٔ دوره‌گرد

- بدین ترتیب می‌توانیم مقدار کران را در هریک از رئوس درخت فضای حالت به صورت زیر محاسبه کنیم.



مسئلهٔ فروشندهٔ دوره‌گرد

- مقدار کران را برای همهٔ رئوس محاسبه می‌کنیم و رأسی را برای بسط دادن انتخاب می‌کنیم که مقدار کران آن کمینه باشد.
- در اینجا از بین رئوس $[1,5]$,$[1,4]$,$[1,3]$,$[1,2]$ رأس $[1,3]$ کوچکترین کران را دارد که برابر با ۲۲ است.
- وقتی رأس $[1,3]$ بسط داده شد، در نهایت کوچکترین طول مسیری که در ریشه به دست می‌آید برابر با ۳۱ است.
- اما در رأس $[1,4]$ همچنان یک کران کوچکتر برابر با ۳۰ وجود دارد، بنابراین آن رأس را بسط می‌دهیم و در نهایت مسیری با طول ۳۰ پیدا می‌کنیم.

- هر رأس در درخت فضای حالت را می‌توانیم با ساختمان زیر نشان دهیم.

Algorithm Node Structure

```
struct node
    int level    ▷ the node's level in the tree
    ordered-set path
    int bound
```

مسئلهٔ فروشندهٔ دوره‌گرد

- در الگوریتم زیر برای یک گراف با n رأس، که وزن یال‌های آن با ماتریس W داده شده است، می‌خواهیم کوتاهترین دور همیلتونی را پیدا کنیم.

Algorithm Travelling Salesman Problem

```
function TRAVELLING-SALESMAN-PROBLEM(int n, int-matrix W)
ordered-set opttour  ▷ Optimal tour
int minlength = ∞  ▷ The length of the optimal tour
priority-queue PQ
node u,v
initialize(PQ)  ▷ Initialize PQ to be empty.
v.level = 0
v.path = [1]  ▷ Make first vertex the
v.bound = bound(v)  ▷ starting one.
insert(PQ,v)
```

مسئلهٔ فروشندهٔ دوره‌گرد

Algorithm Travelling Salesman Problem

```
while !empty(PQ) do
   remove(PQ, v)     ▷ Remove node with best bound.
   if (v.bound < minlength) then
      u.level = v.level + 1        ▷ Set u to a child of v.
      for (all i not in v.path) do
         u.path = v.path + i            ▷ put i at the end of v.path
         ▷ Check if next vertex completes a tour.
         if (u.level == n-2) then
         ▷ put the last vertex not in u.path and also vertex 1 at the end of u.path
            u.path = u.path + last-vertex + 1
            ▷ Function length computes the length of the tour.
            if (length(u) < minlength) then
               minlength = length(u)
               opttour = u.path
         else
            u.bound = bound(u)
            if (u.bound < minlength) then
               insert(PQ, u)
return (opttour, minlength)
```

مسئلهٔ فروشندهٔ دوره‌گرد

- کران یک رأس درخت فضای حالت را به صورت زیر محاسبه می‌کنیم.

Algorithm Computing Bound for Travelling Salesman Problem

```
function BOUND(node v)
bound = 0
for edge (i,j) in v.path do
    bound += length(i,j)
min = ∞
i = last vertex in v.path
for vertex k not in v.path do
    if W[i][k] < min then
        min = W[i][k]
bound += min
for vertex i not in v.path do
    min = W[i][1]
    for vertex k not in v.path do
        if W[i][k] < min then
            min = W[i][k]
    bound += min
return bound
```

مسئلهٔ فروشندهٔ دوره‌گرد

- ممکن است در یک الگوریتم شاخه و کران بتوانیم چندین تابع کران را پیدا کنیم که در این موارد معمولاً یک تابع کران بهتری پیدا می‌کند و در نتیجه به جواب می‌رسد.