راهبرد عقبگرد (Backtracking)

راهبرد عقبگرد را برای حل مسائل را با یک مثال شروع می‌کنیم.

مساله n وزیر (n-Queens) از جمله مسائل کلاسیک در این حوزه است.

هدف در این مساله آن است تا n وزیر را در یک صفحه شطرنج $n \times n$ به گونه‌ای قرار دهیم تا هیچ دو وزیری همدیگر را تهدید نکنند.

بنابراین هیچ دو وزیری در یک سطر، ستون و یا قطر قرار نخواهند گرفت.

راهبرد عقبگرد (Backtracking)

به صورت کلی راهبرد عقبگرد برای حل مسائلی مفید هستند که ....

می‌خواهیم **یک توالی** (sequence) را از ...

مجموعه‌ای مشخص از توالی‌ها به گونه‌ای انتخاب کنیم که ....

توالی انتخاب شده معیارهای مشخصی را دارا باشد.

در مساله n وزیر، **توالی** ....

موقعیتی است که هر وزیر در آن قرار می‌گیرد

مجموعه مشخص، ...

n^2 موقعیتی در صفحه شطرنج است که هر وزیر می‌تواند در آن قرار گیرد. پس مجموعه در این مثال $n^2 \times n^2 \times \ldots \times n^2$ عضو دارد.

معیار نیز آن است که ....

هیچ دو وزیری همدیگر را تهدید نکنند.

راهبرد عقبگرد

عقبگرد، نسخه اصلاح شده‌ای از الگوریتم **پیمایش عمقی درخت** یا ...

Depth First Search (DFS) می‌باشد.

به طور کلی در الگوریتم‌های پیمایش عمقی درخت، از ریشه درخت کار پیمایش شروع می‌شود و ...

تا حد امکان در شاخه‌ها کار پیمایش انجام می‌شود و سپس ...

به ریشه بازگشت انجام می‌شود تا پیمایش در دیگر شاخه‌ها صورت پذیرد

راهبرد عقبگرد

جهت یادآوری: ۳ نوع پیمایش DFS وجود دارد:

Pre-order

ابتدا داده ریشه مشاهده می‌شود (یا المان جاری)
زیردرخت سمت چپ به صورت بازگشتی با همین رویکرد پیمایش می‌شود
زیردرخت سمت راست به صورت بازگشتی با همین رویکرد پیمایش می‌شود

In-order (symmetric)

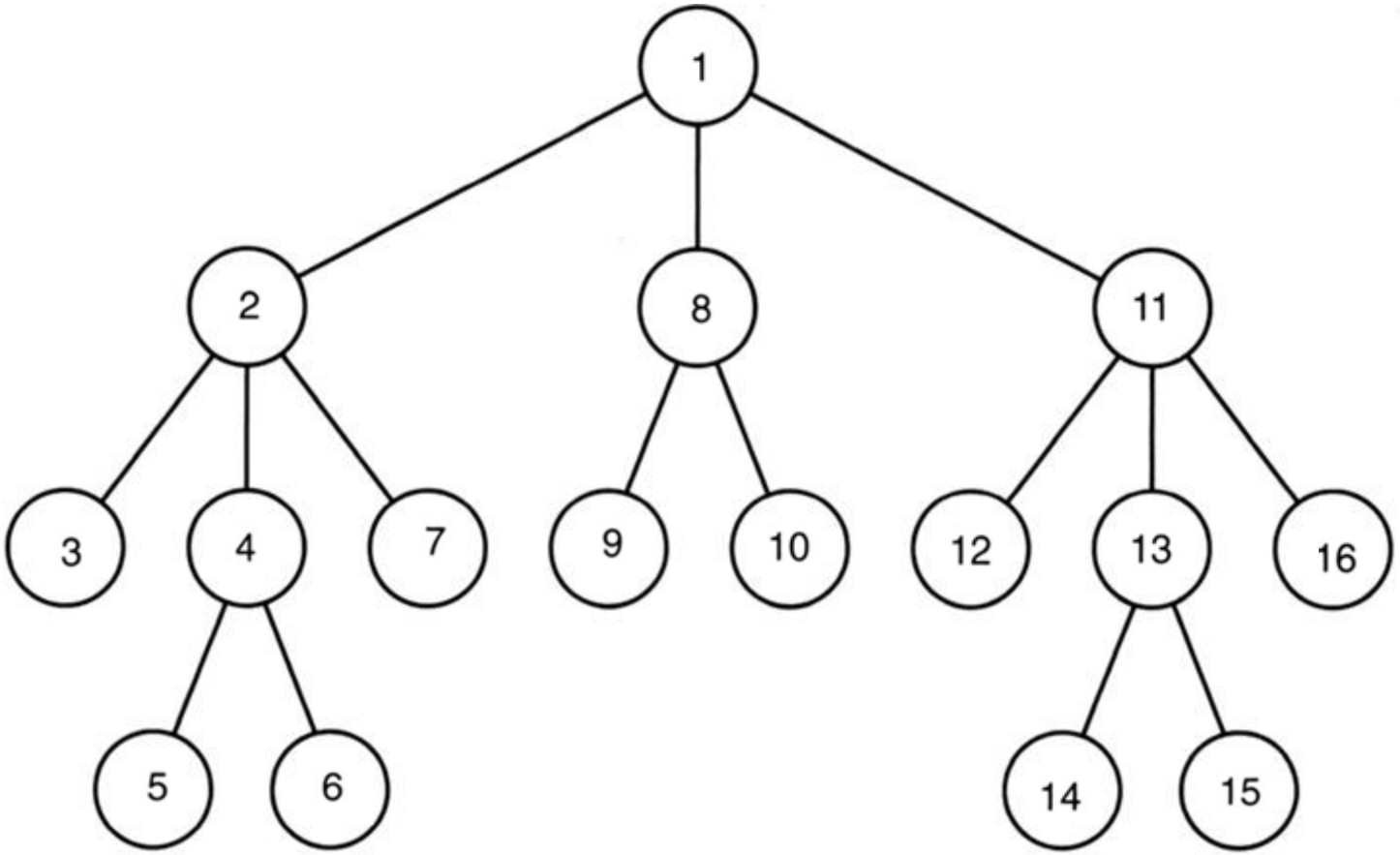
ابتدا زیردرخت سمت چپ به صورت بازگشتی با همین رویکرد پیمایش می‌شود
سپس داده ریشه مشاهده می‌شود (یا المان جاری)
سپس زیردرخت سمت راست به صورت بازگشتی با همین رویکرد پیمایش می‌شود

Post-order

ابتدا زیردرخت سمت چپ به صورت بازگشتی با همین رویکرد پیمایش می‌شود
سپس زیردرخت سمت راست به صورت بازگشتی با همین رویکرد پیمایش می‌شود
سپس داده ریشه مشاهده می‌شود (یا المان جاری)

راهبرد عقبگرد

در ادامه درخت شکل زیر با پیمایش عمقی با رویکرد pre-order ، ...
همان رویکرد راهبرد عقبگرد پیمایش می‌شود.



راهبرد عقبگرد

به مساله n وزیر برمی‌گردیم.
برای n=4، می‌خواهیم تکنیک عقبگرد را استفاده کنیم.
پس وزیرها باید در صفحه شطرنج با ابعاد 4 × 4 قرار گیرند به گونه‌ایکه ...
هیچ دو وزیری همدیگر را تهدید نکنند.
در ابتدای کار برای چیدن وزیرها بی‌درنگ این مطلب به ذهن می‌رسد که ...
هیچ دو وزیری نمی‌توانند در یک سطر قرار گیرند.
بنابراین برای حل توالی خاصی که به دنبالش هستیم، اینگونه عمل می‌کنیم که ...
هر وزیر را به سطر مشخصی تخصیص می‌دهیم و کنترل می‌کنیم که ...
چه ستونی برای هر کدام سبب حل مساله می‌شود.
در این مساله باتوجه به اینکه هر وزیر تنها می‌تواند در یکی از چهار ستون قرار گیرد ...
بنابراین مجموعه توالی‌ها به تعداد ...
$4 \times 4 \times 4 \times 4 = 256$ توالی کاندید دارد.

راهبرد عقبگرد

به صورت زیر می‌توانیم راه حل‌های کاندید را ایجاد کنیم ...

درختی ایجاد می‌کنیم که ...

انتخاب ستونی برای وزیر اول (وزیر اول در کدام ستون قرار گیرد) در گره‌های سطح یک درخت باشد

پس، ریشه چهار فرزند دارد

که برای ریشه و همه گره‌های میانی در این درخت، فرزندانشان را از چپ به راست با ۱ تا ۴ شماره‌گزاری می‌کنیم.

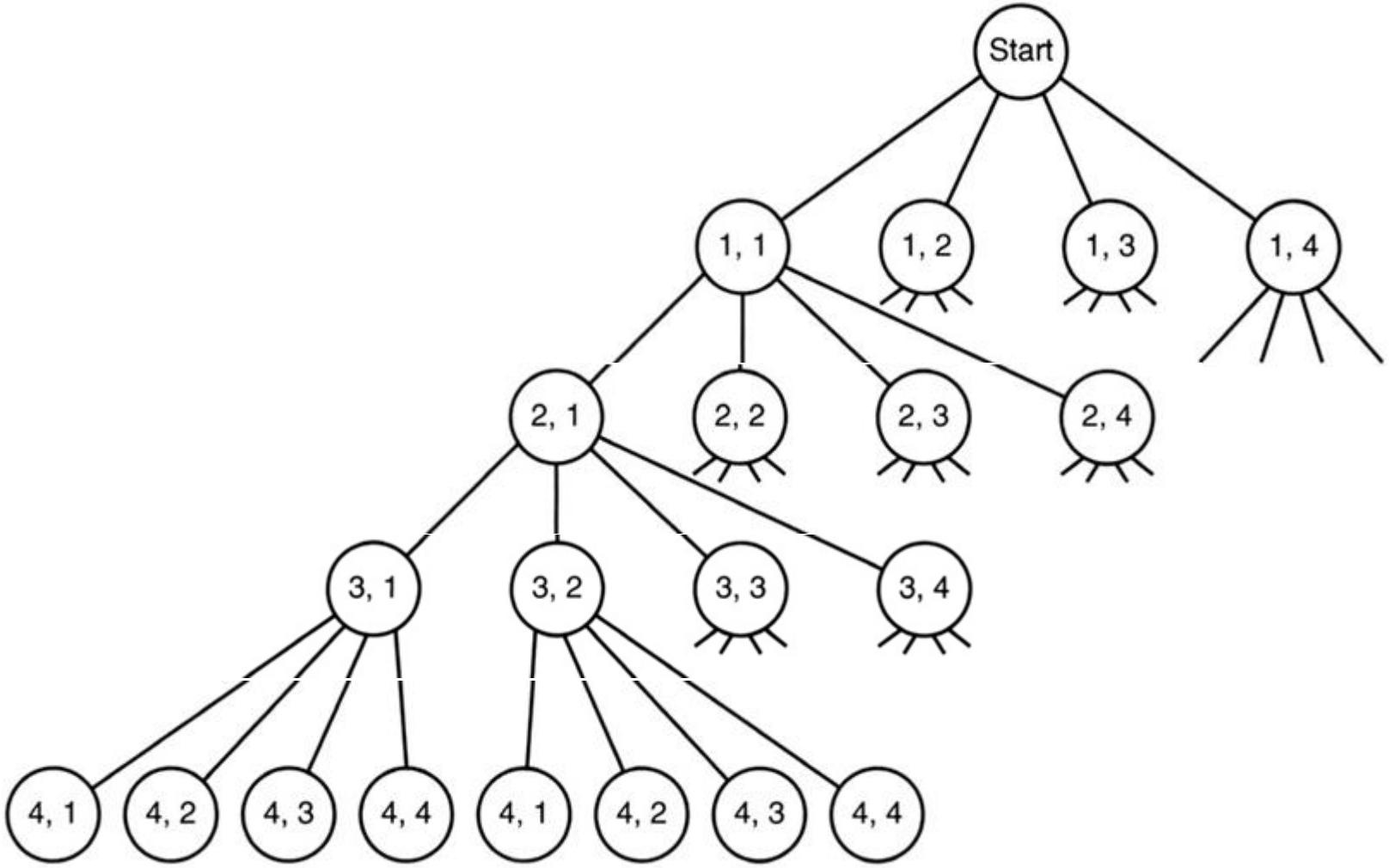
به همین ترتیب ...

هر گره در سطح اول خود چهار فرزند دارد که ...

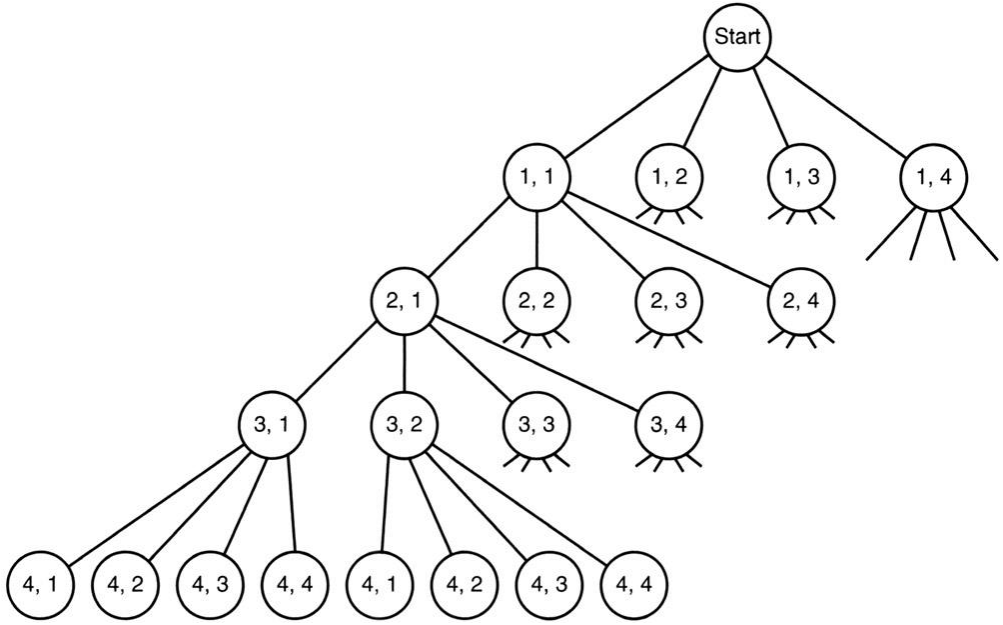
ستون انتخابی برای وزیر دوم در گره‌های این سطح (سطح دوم) قرار دارد.

این روند به همین ترتیب ادامه می‌یابد.

راهبرد عقبگرد



راهبرد عقبگرد



هر مسیری از ریشه به برگ یک راه حل (دنباله موردنظر) می‌تواند باشد.

درختی که با این تفصیل ساخته شد، درخت فضای حالت (state space tree) نامیده می‌شود.

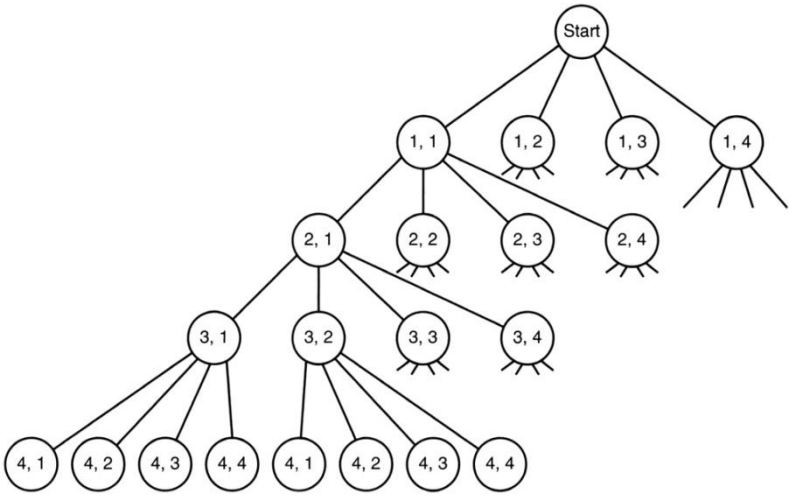
به راحتی می‌توان فهمید که تعداد برگهای این درخت ....

۲۵۶ برگ می‌باشد که هر برگ متناظر با ....

یک راه حل کاندید می‌باشد.

در درخت فضای حالتی که در بالا مشاهده می‌شود، در هر گره زوج مرتب <i,j> برچسب خورده است که به این معنا است که ...

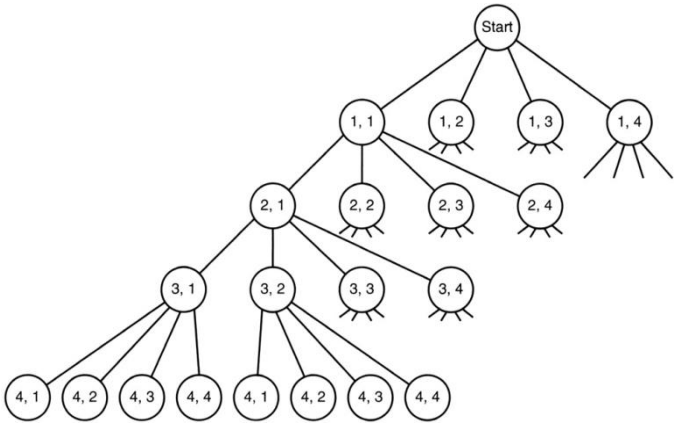
وزیر i ام در ستون j ام قرارگرفته است.



راهبرد عقبگرد

برای آنکه راه حل‌ها را پیدا کنیم، ...
هر راه حل کاندید (مسیر از ریشه به هر برگ) را چک می‌کنیم.
این چک کردن با پیمایش pre-order انجام می‌پذیرد.

$[< 1, 1 >, < 2, 1 >, < 3, 1 >, < 4, 1 >]$
$[< 1, 1 >, < 2, 1 >, < 3, 1 >, < 4, 2 >]$
$[< 1, 1 >, < 2, 1 >, < 3, 1 >, < 4, 3 >]$
$[< 1, 1 >, < 2, 1 >, < 3, 1 >, < 4, 4 >]$
$[< 1, 1 >, < 2, 1 >, < 3, 2 >, < 4, 1 >]$



راهبرد عقبگرد

پیمایشی که در اسلاید قبل آمده است، پیمایشی بسیار ساده است.

این پیمایش به علائمی که در طول مسیر وجود دارد توجهی نمی‌کند.

با توجه به علایم می‌توان به جستجوی کاراتری رسید.

علایم می‌توانند به ما بگویند که ادامه پیمایش این گره چیزی جز بن‌بست در پی نخواهد داشت.

راهبرد عقبگرد

در راهبرد عقبگرد، روند به این صورت است که ...

بعد از اینکه متوجه شدیم که ادامه پیمایش یک گره چیزی جز بن‌بست در پی نخواهد داشت، ...

به پدر آن گره بر می‌گردیم (**عقبگرد**) و پیمایش را با فزندان بعدی آن ادامه می‌دهیم.

گره‌ای را **نامیدبخش (non-promise)** می‌نامیم که ...

بعد از مشاهده آن تعیین کنیم که ادامه پیمایش آن ما را به راه حل نمی‌رساند.

هر گره‌ای که شرط بالا را نداشته باشد را **امیدبخش (promise)** می‌نامیم.

راهبرد عقبگرد

بنابراین **راهبرد عقب‌گرد** یعنی ...

برای حل مساله درخت فضای حالت آن را ایجاد کنیم

سپس جستجوی عمقی با رویکرد pre-order را در درخت فضای حالت انجام دهیم.

در هر گام از جستجو مشخص کنیم که آیا گره‌ای که الان در حال مشاهده آن هستیم، امید بخش هست یا نه ...

چنانچه امیدبخش باشد، کار را ادامه می‌می‌دهیم و ....

چنانچه امیدبخش نباشد، به گره پدر آن برمی‌گردیم.

این که با امیدبخش نبودن گره‌ای به پدر آن برگردیم و عملا گره‌های فرزند آن را پیمایش نکنیم، به اصطلاح درخت در آن گره به بعد هرس(prune) می‌شود و ...

درخت فضای حالت پیمایش شده با این رویکرد را ...

درخت فضای حالت هرس شده (pruned state space tree) نامیده می‌شود.

راهبرد عقبگرد

الگوریتم عمومی رویکر عقبگرد به صورت زیر می‌باشد.

```
function checknode (v)
{
    if (promising(v))
        if (there is a solution at v)
            write the solution;
        else
            for (each child u of v)
                checknode(u);
}
```

راهبرد عقبگرد

```
function checknode (v)
{
        if (promising(v))
                if (there is a solution at v)
                  write the solution;
                else
                  for (each child u of v)
                    checknode(u);
 }
```

در اولین گام، ریشه درخت به تابع *checknode* به عنوان پارامتر ارسال می‌شود.

وقتی تابع *checknode* با گره‌ای صدا زده می‌شود (گره در پیمایش مشاهده می‌شود) ...

ابتدا چک می‌شود که آیا امیدبخش است یا خیر

چنانچه امیدبخش بود و راه‌حلی در آن گره وجود داشته باشد، ...

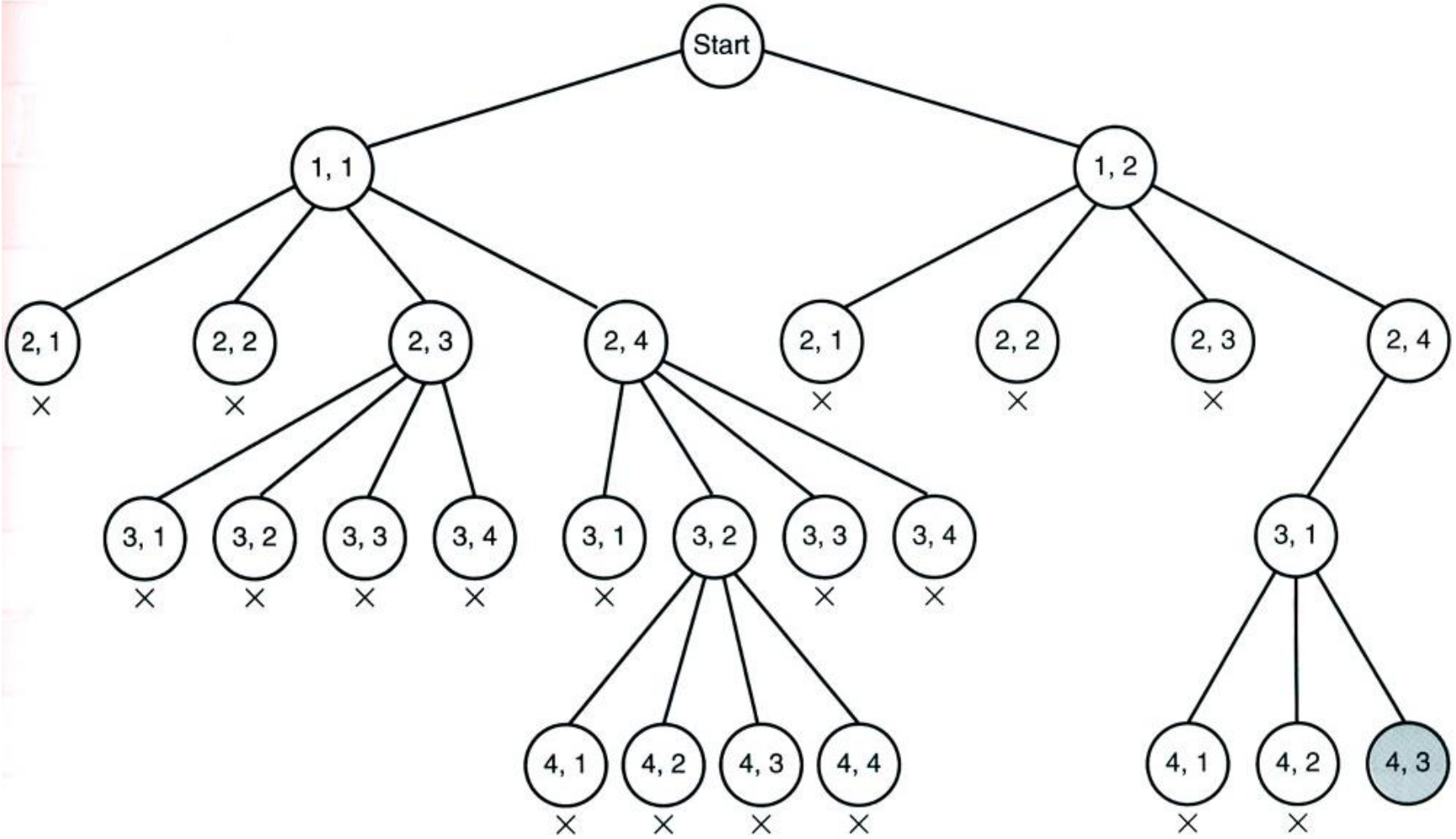
راه حل چاپ می‌شود.

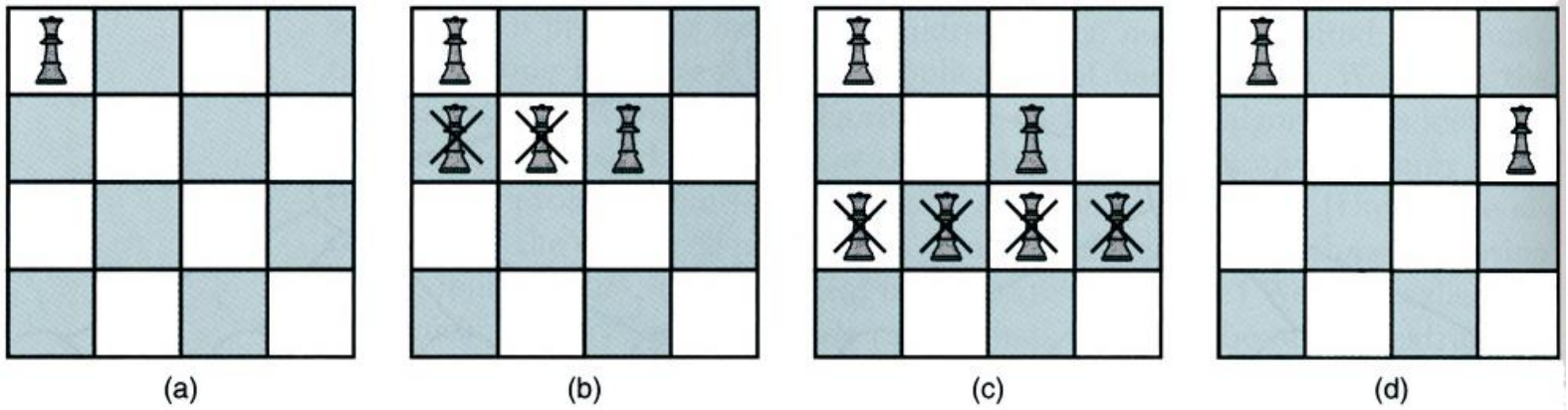
اگر راه حل در گره امیدبخشی وجود نداشت، ....

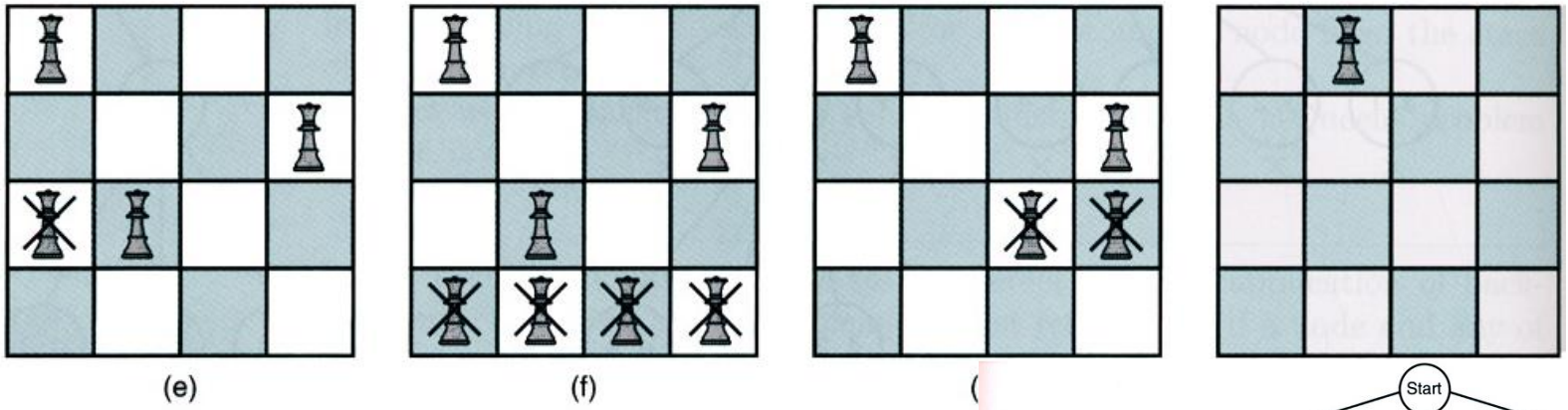
تمامی فرزندان آن صدا زده می‌شوند.

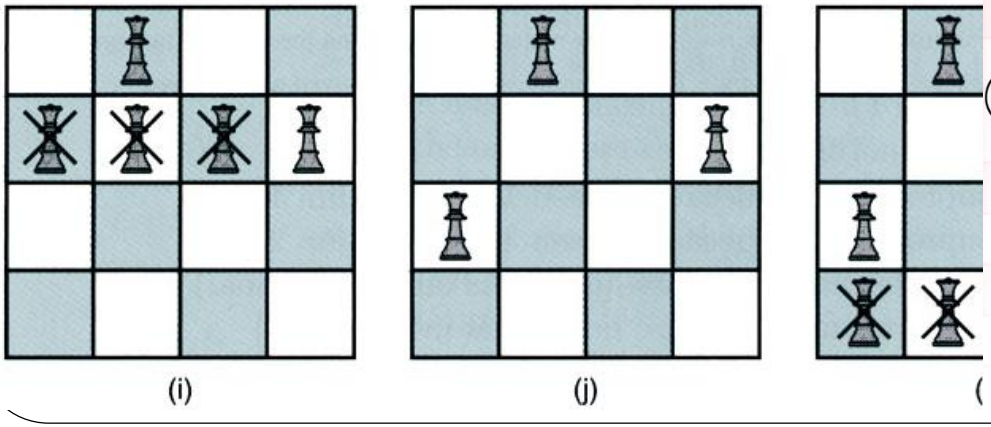
بسته به هر مساله، تابع promising متفاوت است که بایستی برای آن مساله پیاده‌سازی شود.

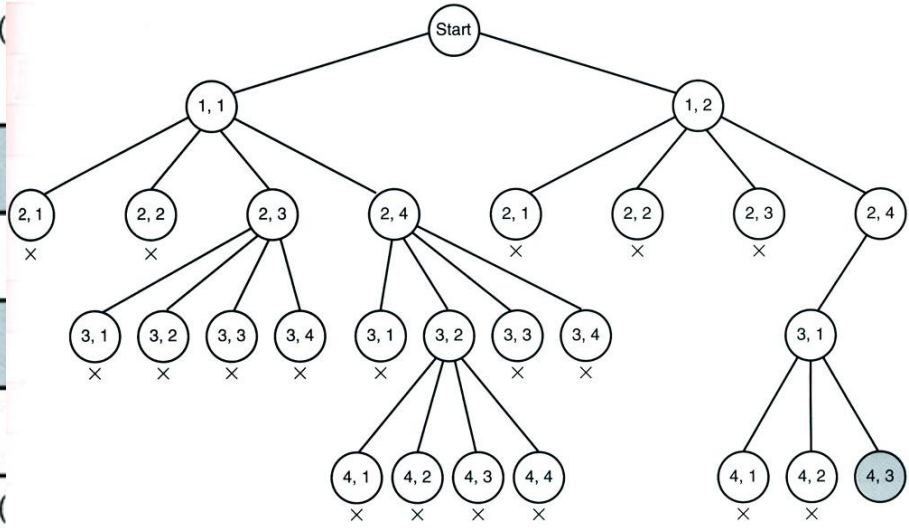
راهبرد عقبگرد




(a)
(b)
(c)
(d)


(e)
(f)


(i)
(j)

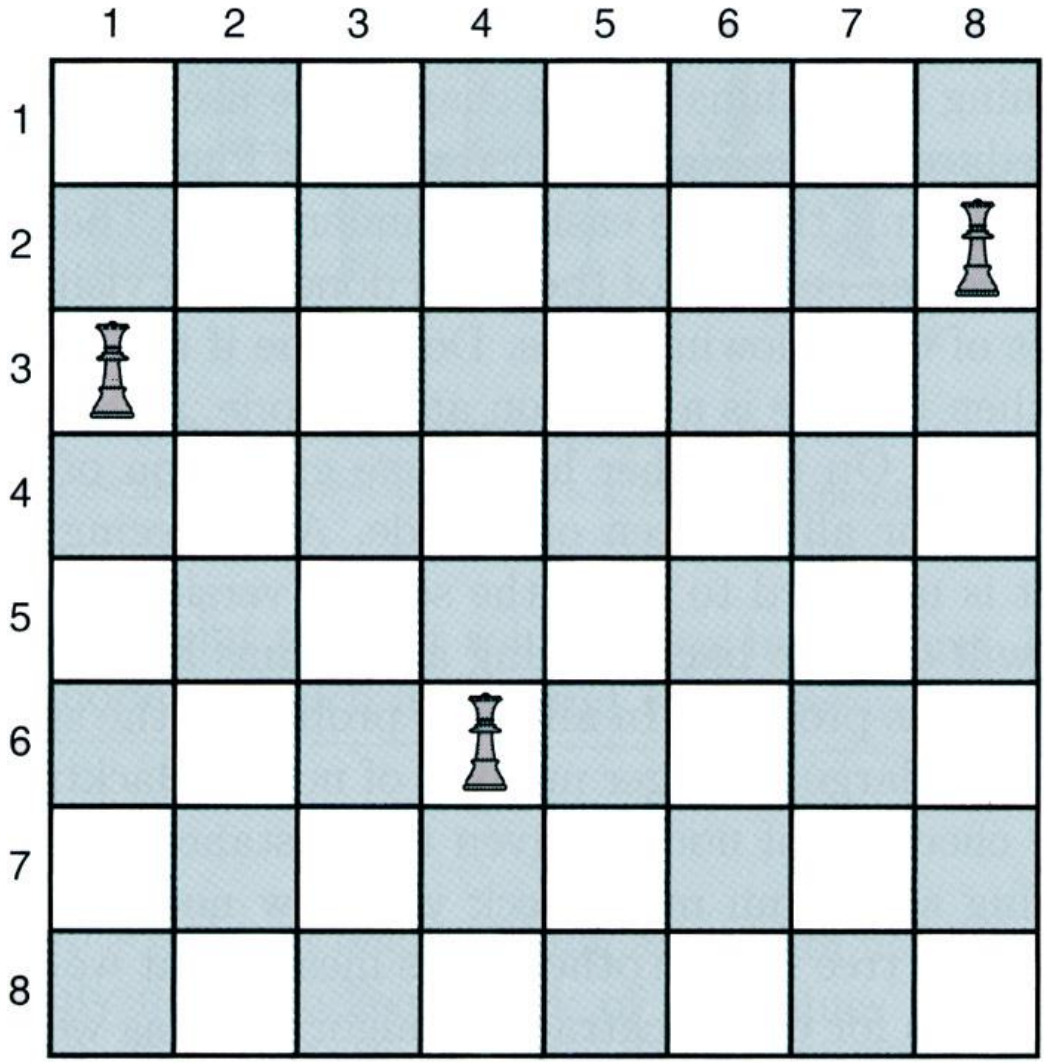

Start
1, 1
1, 2
2, 1
2, 2
2, 3
2, 4
2, 1
2, 2
2, 3
2, 4
3, 1
3, 2
3, 3
3, 4
3, 1
3, 2
3, 3
3, 4
3, 1
4, 1
4, 2
4, 3
4, 4
4, 1
4, 2
4, 3

راهبرد عقبگرد- مسئله n وزیر

در مساله n وزیر، تابع promising بایستی چک کند که آیا دو وزیر در ستون یا قطر یکسانی قرار دارند یا خیر.

چنانچه col(i) بیانگر ستونی باشد که وزیر در سطر iام در آن قرار دارد ...

بنابراین برای چک‌کردن اینکه وزیر سطر kام، در ستون یا قطر یکسانی با وزیر سطر iام قرار دارد یا خیر، می‌بایست روابط زیر را کنترل کنیم:



راهبرد عقبگرد

$$col\,(i) = col\,(k)\,.$$

$$col\,(i) - col\,(k) = k - i$$

$$col\,(i) - col\,(k) = i - k$$

راهبرد عقبگرد

```
function r=promising(i)
   r=1;
   for k=1:i-1
      if (col(i)==col(k))||(col(i)-col(k)==i-k)||(col(i)-col(k)==k-i)
         r=0;
         break;
      end
   end
end
```

راهبرد عقبگرد

```
function queens(i)
   if promising(i)
      if(i==n)
         disp(col);
      else
         for j=1:n
            col(i+1)=j;
            queens(i+1);
         end
      end
   end
end
```

مسئله حاصل‌جمع زیر مجموعه‌ها

در این مساله،

n عدد صحیح مثبت $(w_i, i=1,2,\ldots,n)$ وجود دارد

همچنین عدد صحیح مثبت W را داریم

هدف آن است تا تمامی زیرمجموعه‌هایی از اعداد صحیح را پیدا کنیم که مجموع آنها W می‌شود.

مسئله حاصل‌جمع زیر مجموعه‌ها

در این مساله هم همانند مساله n وزیر به دنبال آن هستیم تا تمامی راه حل‌ها ممکن را پیدا کنیم

مسئله حاصل‌جمع زیر مجموعه‌ها

برای n=5 و W=21

$$w_1 = 5 \qquad w_2 = 6 \qquad w_3 = 10 \qquad w_4 = 11 \qquad w_5 = 16.$$

داریم:

$$w_1 + w_2 + w_3 = 5 + 6 + 10 = 21,$$
$$w_1 + w_5 = 5 + 16 = 21, \text{and}$$
$$w_3 + w_4 = 10 + 11 = 21,$$

بنابراین پاسخ‌های مساله به صورت سه مجموعه

$$\{w_1, w_2, w_3\}, \{w_1, w_5\}, \text{and } \{w_3, w_4\}$$

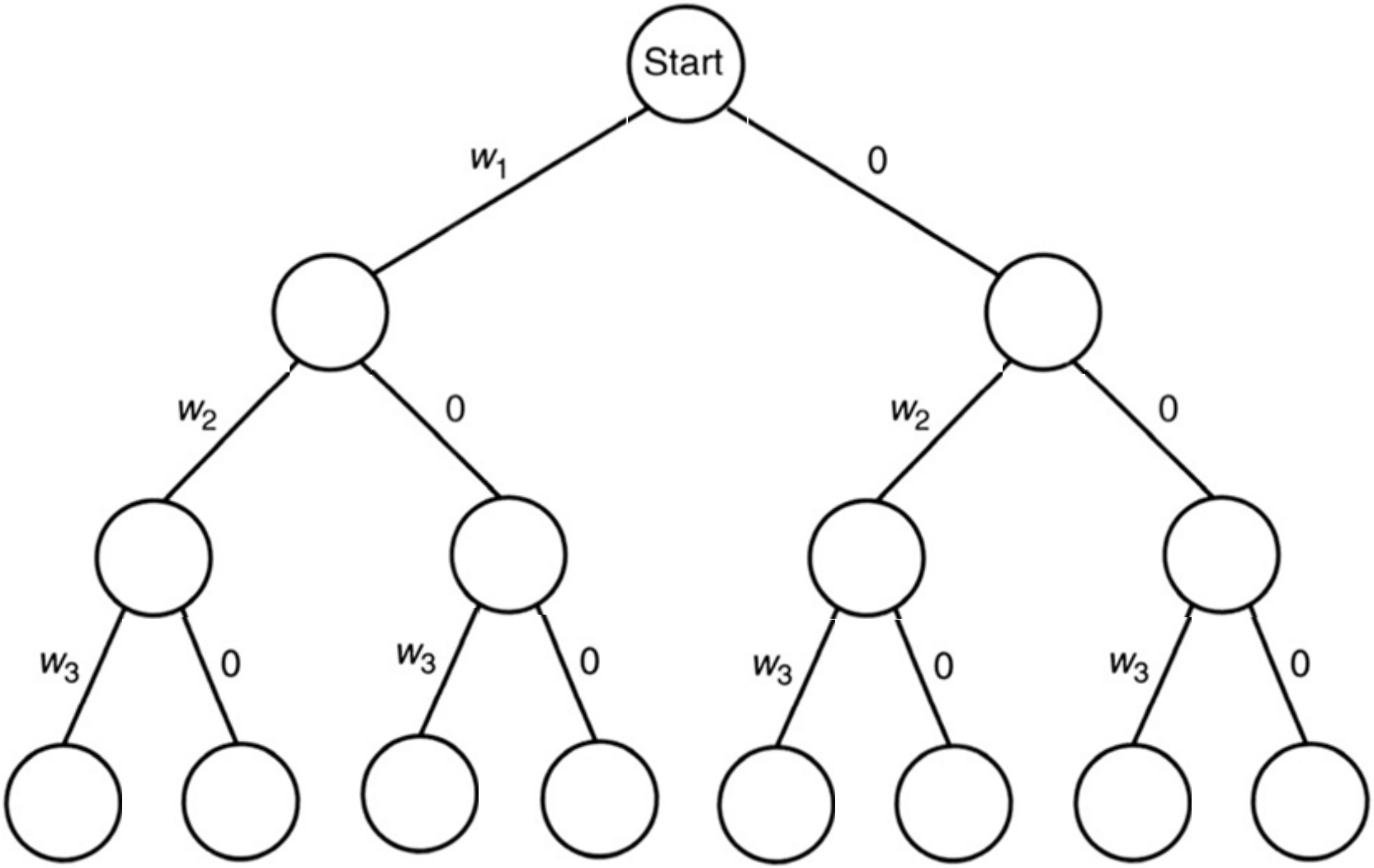
مسئله حاصل‌جمع زیر مجموعه‌ها

برای مثال داده‌شده در اسلاید قبل، حل مساله با یک بررسی ساده می‌تواند انجام شود.

برای nهای بزرگتر نیاز است تا به یک راه‌حل سیستماتیک برسیم.

یک رویکرد آن است تا درخت فضای حالت آن را ایجاد کنیم.

مسئله حاصل‌جمع زیر مجموعه‌ها



مسئله حاصل‌جمع زیر مجموعه‌ها

در هر گره، مجموع وزن‌هایی که تا آن گره به مجموعه اضافه‌شده‌اند را می‌نویسیم.

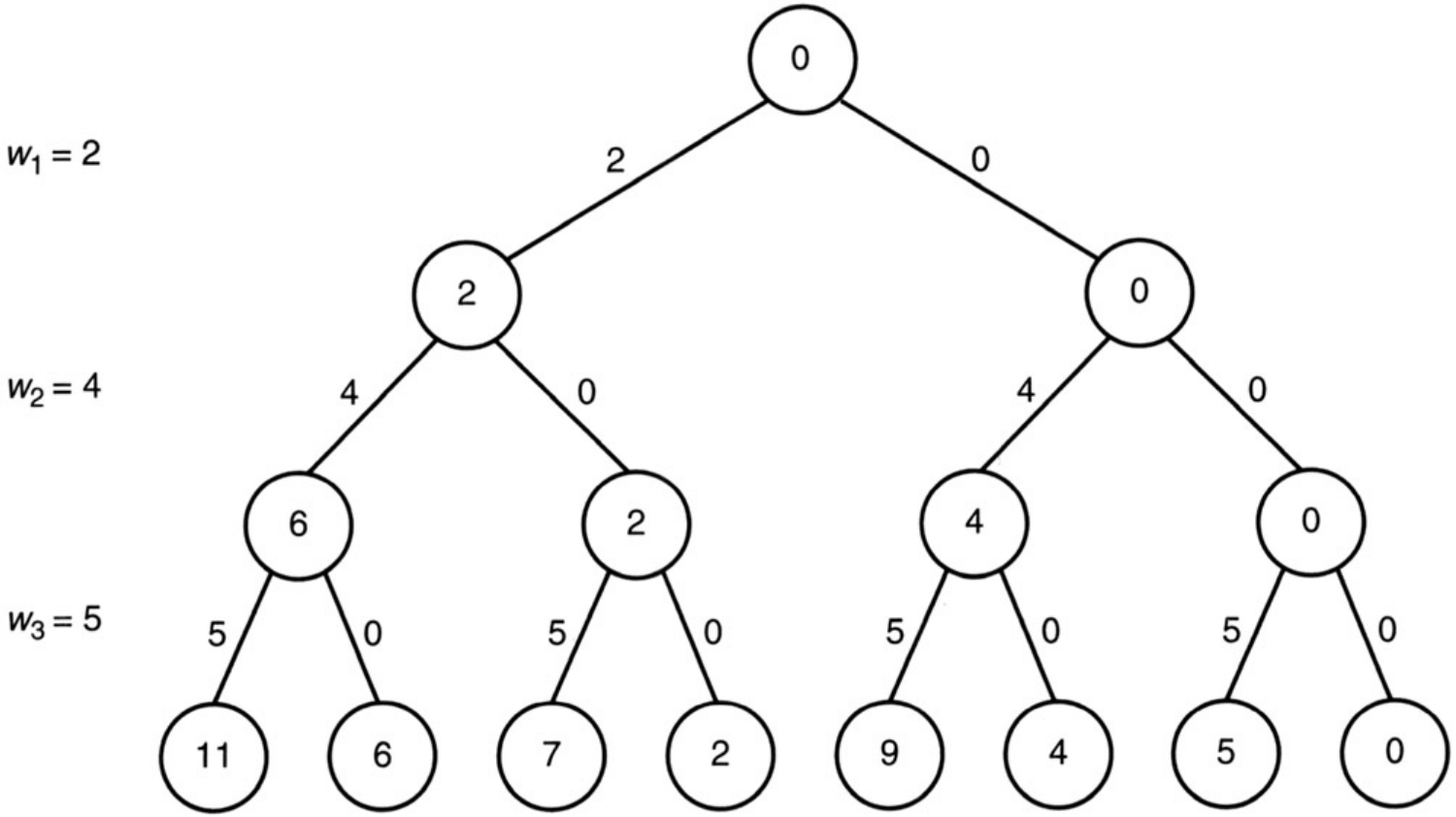
بنابراین هر برگ دربردارنده ...

مجموع وزن‌هایی می‌شود که در زیرمجموعه‌ای قراردارند که به آن برگ منتهی می‌شوند.

مسئله حاصل‌جمع زیر مجموعه‌ها

درخت فضای حالت برای n=3 و W=6 به صورت زیر می‌باشد:

$$w_1 = 2 \qquad w_2 = 4 \qquad w_3 = 5.$$



مسئله حاصل‌جمع زیر مجموعه‌ها

چنانچه قبل از جستجو وزن‌ها را به صورت صعودی مرتب کنیم

آیا تفاوتی در پاسخ‌های دزحت فضای حالت ایجاد می‌شود؟

ولی آیا با این رویکرد علائمی پدیدار می‌شوند که درباره ناامیدبخش بودن گره‌ها به ما هشدار دهند؟

مسئله حاصل‌جمع زیر مجموعه‌ها

چنانچه وزن‌ها را به صورت صعودی مرتب کنیم، ...

سبکترین وزن باقی‌مانده پس از آنکه ما در سطح iام هستیم ... w_{i+1} می‌باشد.

چنانچه wight برابر با مجموع وزن‌هایی باشد که تا آن گره (که در سطح i قراردارد) در مجموعه شامل شده‌اند ....

مسئله حاصل‌جمع زیر مجموعه‌ها

گره موجود در سطح i اگر weight اش برابر با W شود که یک راه حل است ولی....

ناامیدبخش است اگر ...

$$weight + w_{i+1} > W.$$

مسئله حاصل‌جمع زیر مجموعه‌ها

همچنین غیر از weight متغیر دیگری با نام total برای هر گره درنظر می‌گیریم ...

که برابر است با مجموع وزن‌های باقی‌مانده در سطوح بعدی گره جاری است.

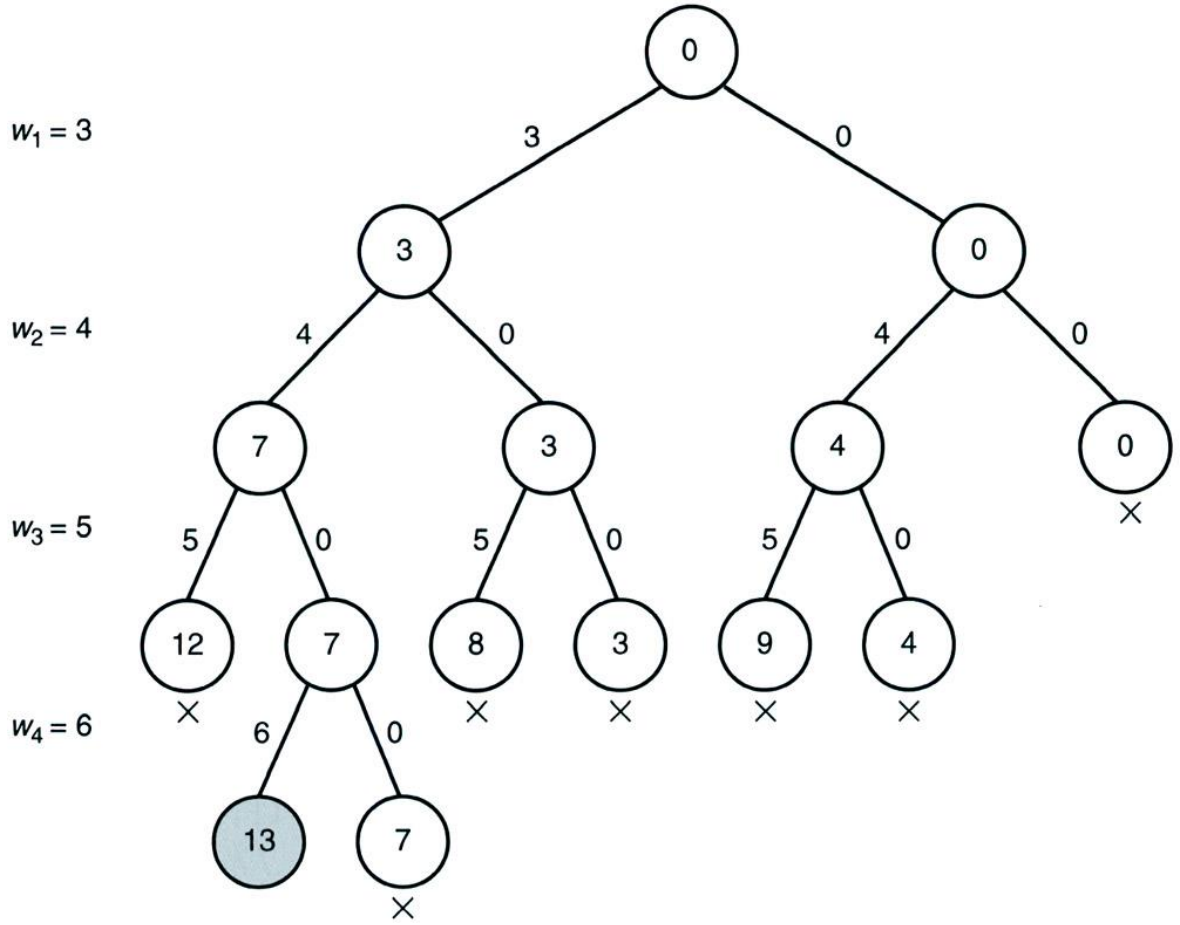
بنابراین علامت دیگری که درباره ناامیدبخش بودن یک گره می‌توان استنتاج کرد ....

$$weight + total < W.$$

مسئله حاصل‌جمع زیر مجموعه‌ها

$n = 4,\ W = 13,$

$w_1 = 3 \qquad w_2 = 4 \qquad w_3 = 5 \qquad w_4 = 6.$



مسئله حاصل‌جمع زیر مجموعه‌ها

```
function r=promising_sum_of_subsets(i,weight,total)
    global W;
    global w;
    if(weight+total>=W)&&(wight==W || weight+w(i+1)<=W)
        r=1;
    else
        r=0;
    end
end
```

مسئله حاصل‌جمع زیر مجموعه‌ها

```
function sum_of_subsets(i,weight,total)
    global W;
    global w;
    if promising_sum_of_subsets(i,weight,total)
        if (weight==W)
            disp(include(1:i))
        else
            include(i+1)=true;
            sum_of_subsets(i+1,weight+w(i+1),total-w(i+1));
            include(i+1)=false;
            sum_of_subsets(i+1,wight,total-w(i+1));
        end
    end
end
```

مسئله حاصل‌جمع زیر مجموعه‌ها

برای شروع پیمایش درخت فضای حالت را اینگونه صدا می‌زنیم:

Sum_of_subsets(0,0,sum(w))

تعداد کل گره‌هایی که در درخت فضای حالت وجود دارد برابر است با:

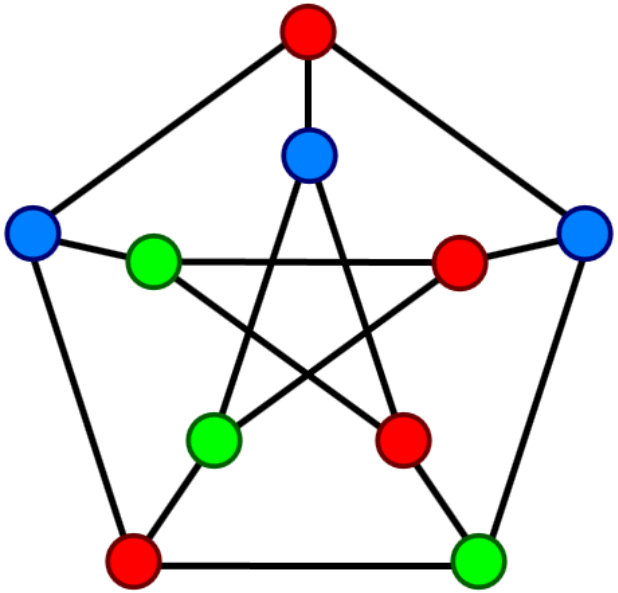
$$1 + 2 + 2^2 + \cdots + 2^n = 2^{n+1} - 1.$$

رنگ‌آمیزی گراف

در این مساله که با عنوان m-Coloring شناخته می‌شود ...

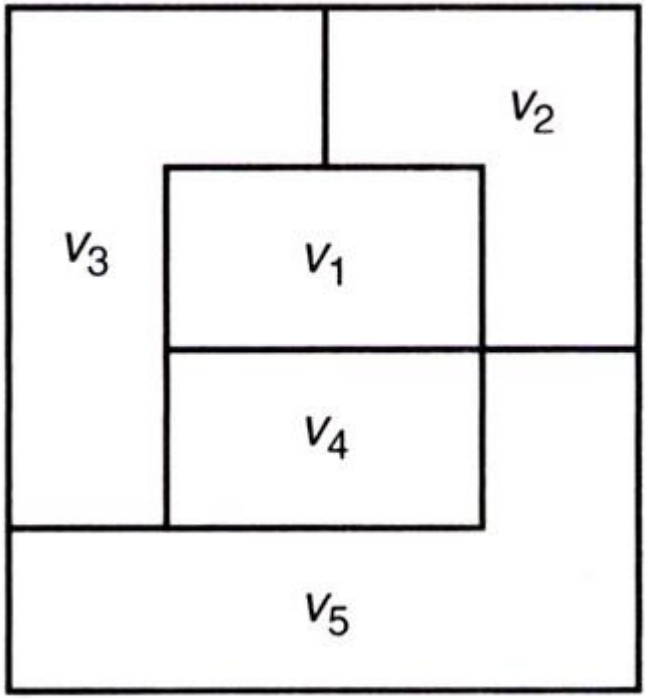
به دنبال یافتن راه‌هایی هستیم تا راس‌های گرافی غیر جهت‌دار را با حداکثر m رنگ به گونه‌ای رنگ‌آمیزی کنیم تا ...

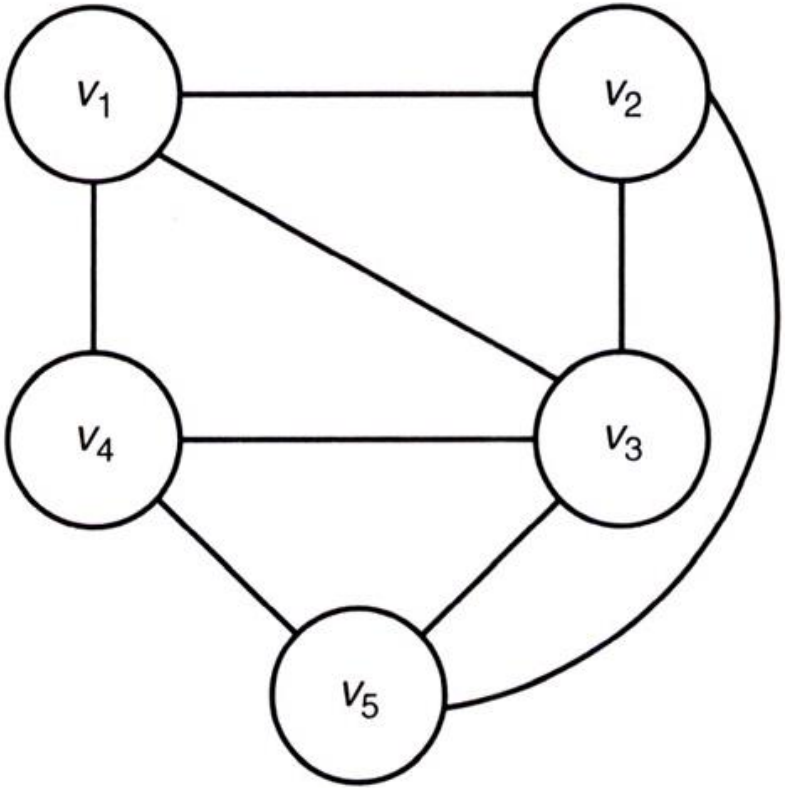
هیچ دو راس مجاوری رنگ یکسانی نداشته باشند.



رنگ‌آمیزی گراف

یکی از کاربردهای این مساله رنگ‌آمیزی نقشه‌ها است.





رنگ‌آمیزی گراف

چگونه درخت فضای حالت را ایجاد کنیم؟

در سطح ۱ تمامی رنگ‌های ممکن برای v_1 درنظر گرفته می‌شود.

در سطح ۲، به ازای هر رنگ v_1، تمامی رنگ‌های ممکن برای v_2 درنظر گرفته می‌شود.

و به همین ترتیب تا ...

تمامی رنگ‌های ممکن برای v_n در سطح nام درنظر گرفته می‌شود.

رنگ‌آمیزی گراف

در این درخت فضای حالت، هر مسیری از ریشه به برگ یک راه حل کاندید می‌تواند باشد.

با چک کردن اینکه هیچ دو گره مجاور در هر راه حل کاندید دارای رنگ مشابهی نباشند می‌توان پاسخ بودن هر راه‌حل کاندید را آزمون کرد.

برای اینکه مشخص باشد که درباره درخت یا گراف صحبت می‌کنیم ...

گره (node) به درخت فضای حالت اشاره می‌کند و

راس (vertex) به گرافی اشاره می‌کند که تصمیم به رنگ‌آمیزی آن داریم.

رنگ‌آمیزی گراف

آیا از رویکرد عقبگرد در این مساله می‌توانیم استفاده کنیم؟

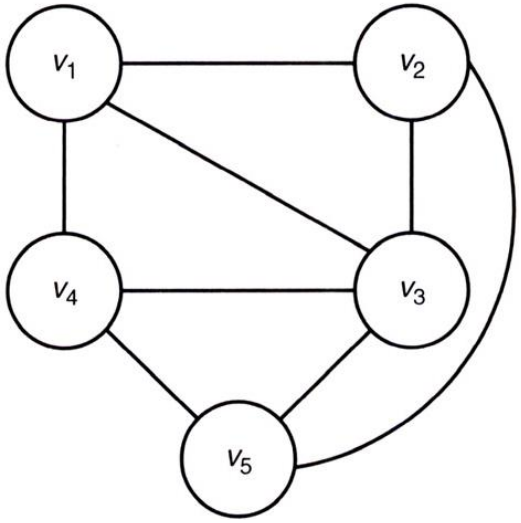
آیا شرطی برای ناامیدبخش بودن هر گره می‌توان پیدا کرد یا باید تمای درخت فضای حالت را برای بدست‌آوردن پاسخ‌های مساله پیمایش کرد؟

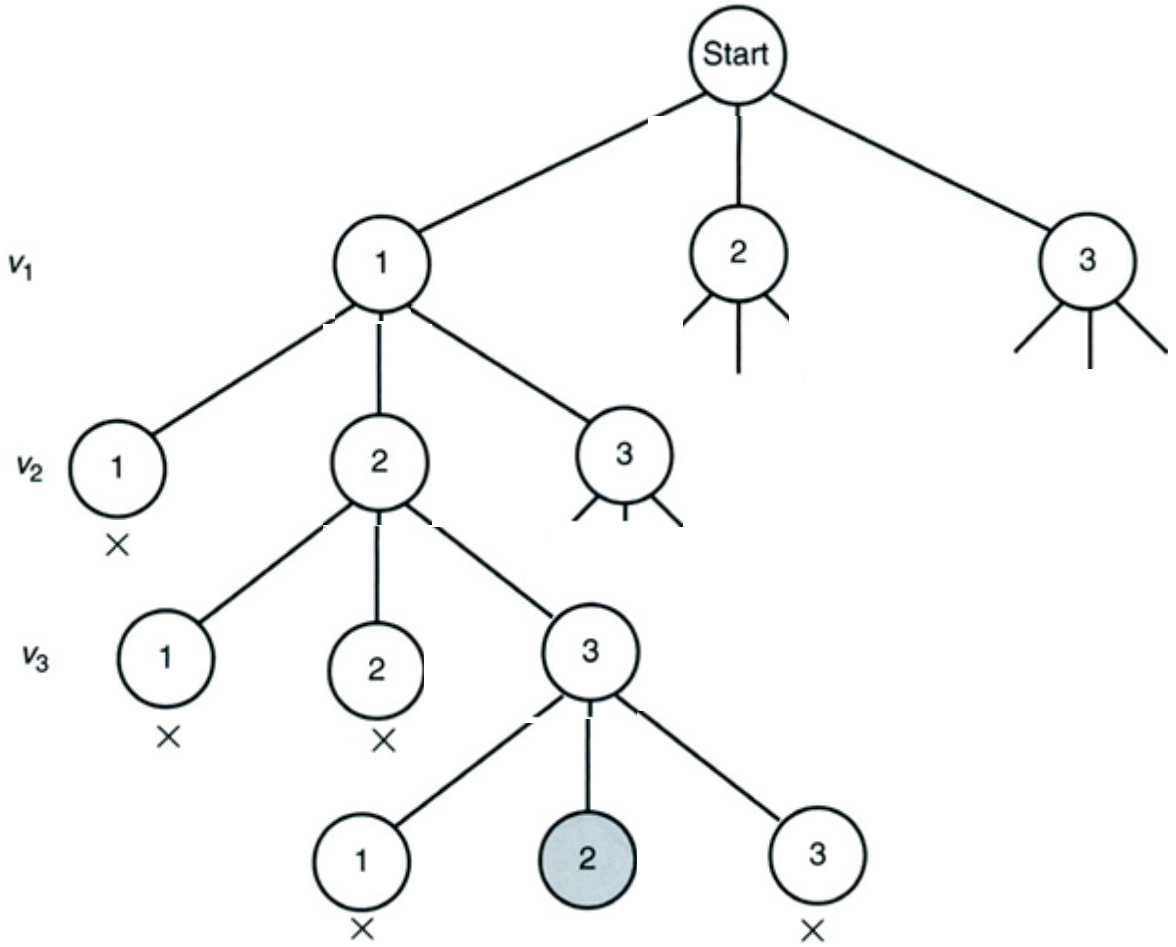
برای راس متناظر با هر گره می‌توان چک کرد که آیا ...

همرنگ راس‌هایی که قبلا رنگ شده‌اند و مجاورشان نیز می‌باشد هست یا خیر؟

رنگ‌آمیزی گراف

حل مساله 3-Coloring را با روش عقبگرد برای مثال زیر گام به گام نشان دهید.





رنگ‌آمیزی گراف

```
function m_coloring(i)
    if m_coloring_promising(i)
        if (i==n)
            disp(vcolor)
        else
            for color=1:m
                vcolor(i+1)=color;
                m_coloring(i+1);
            end
        end
    end
end
```

رنگ‌آمیزی گراف

```
function m_coloring_promising(i)
    r=true;
    j=1;
    while(j<i && r)
        if(W(i,j)&&(vcolor(i)==vcolor(j)))
            r=false;
        end
        j=j+1;
    end
end
```

رنگ‌آمیزی گراف

تعداد گره‌های درخت فضای حالت در این الگوریتم برابر است با:

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

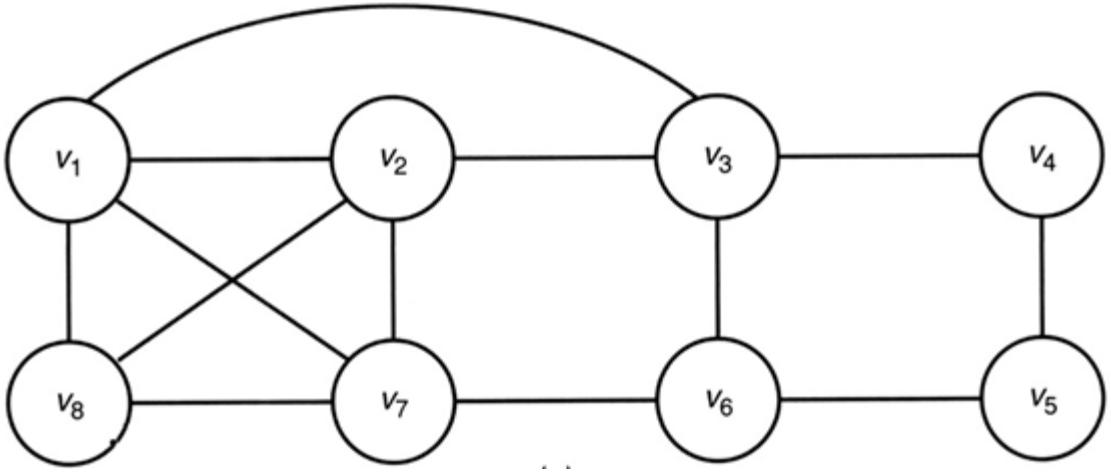
مسئله مدارهای هامیلتونی

گرافی متصل و غیرجهت‌دار مفروض می‌باشد ...

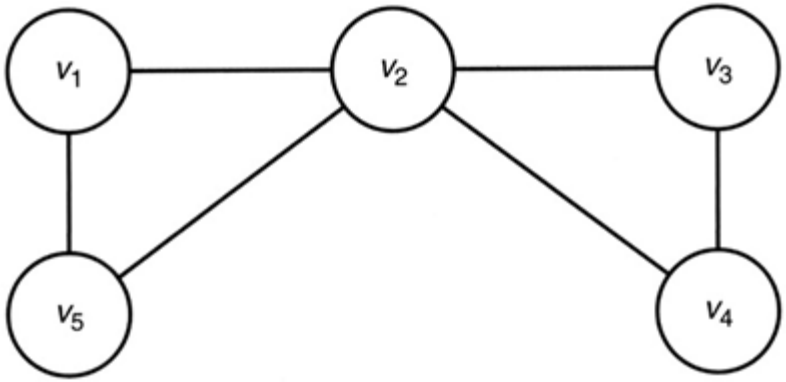
مدار هامیلتونی (تور) مسیری است که

از یکی از راس‌های گراف شروع شود و از هر راس **دقیقا یکبار بگذرد** و در پایان به راس شروع برگردد.

مسئله مدارهای هامیلتونی



مدار هامیلتونی $[v_1, v_2, v_8, v_7, v_6, v_5, v_4, v_3, v_1]$



شامل هیچ مدار هامیلتونی نمی‌شود

مسئله مدارهای هامیلتونی

در این مساله درخت فضای حالت به صورت زیر متصور است:

فرض کنیم راس شروع حرکت، گره ریشه درخت باشد.

در سطح ۱، هر راس دیگر به غیر از راس شروع به عنوان اولین راس در مسیر حرکت می‌تواند قرار گیرد.

در سطح ۲، بازهم راس‌هایی که در سطح ۱ قرارگرفتند، قرار گیرند.

به همین ترتیب در سطح ...

در سطح n-1، بازهم تمامی n-1 راس غیر از راس شروع قرار گیرند.

مسئله مدارهای هامیلتونی

معیارهای امیدبخش بودن هر گره در گراف چیست؟

یا به عبارتی دیگر چه معیارهایی را می‌توان درنظر گرفت که در صورت برقرار نبودن می‌توان درخت فضای حالت را از آن گره به بعد هرس کرد؟

الف) iامین راس در مسیر باید در گراف همجوار با (i-1)امین راس در مسیر باشد.

ب) به همین ترتیب، (n-1)امین راس در مسیر باید همسایه راس شروع در مسیر باشد.

ج) iامین راس در مسیر نباید هیچکدام از i-1 راس قبلی باشد.

مسئله مدارهای هامیلتونی

```
function hamiltonian(i)
    if promising(i)
        if (i==n)
            disp(vindex)
        else
            for j=2:n
                vindex(i+1)=j;
                hamiltonian(i+1);
            end
        end
    end
end
```

```
function r=hamiltonian_promising(i)
    r=true;
    if (i>1 && ~W(vindex(i-1),vindex(i)) )
        r=false;
    else
        if (i==n && ~W(vindex(n),vindex(1)))
            r=false;
        else
            for j=1:i-1
                if vindex(i)==vindex(j)
                    r=false;
                    break;
                end
            end
        end
    end
end
```

The *i*th vertex on the path must be adjacent to the (*i* - 1)st vertex on the path.

The (n - 1)st vertex must be adjacent to the 0th vertex (the starting one)

The ith vertex cannot be one of the first i - 1 vertices

مسئله مدارهای هامیلتونی

```
vindex(1)=1;
hmailtonian(1);
```

مسئله مدارهای هامیلتونی

تعداد گره‌ها در درخت فضای حالت؟

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

تحلیل پیچیدگی زمانی الگوریتم‌های عقبگرد

به مساله n وزیر برگردیم ...

می‌خواهیم پیچیدگی زمانی آن را بررسی کنیم.

یعنی عملا باید بررسی کنیم که چه تعداد گره در درخت فضای حالت پیمایش شده‌اند.

اگر بتوانیم آنها را به صورت تابعی از n بیان کنیم، در این صورت توانسته‌ایم تحلیلی از پیچیدگی محاسباتی ارائه دهیم.

تحلیل پیچیدگی زمانی الگوریتم‌های عقبگرد

رویکرد اول:

می‌توانیم سقف تعداد گره‌های مشاهدشده را با ...

شمردن تعداد گره‌های درخت فضای حالت مشخص کنیم.

در درخت فضای حالت مربوط به این مساله داشتیم:

۱ گره در سطح ۰

n گره در سطح ۱

n^2 گره در سطح ۲

...

n^n گره در سطح n

که مجموع تعداد گره‌ها برابر است با:

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

تحلیل پیچیدگی زمانی الگوریتم‌های عقبگرد

به عنوان مثال، برای n=8، درخت فای حالت شامل ....

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

تحلیل پیچیدگی زمانی الگوریتم‌های عقبگرد

رویکرد دوم)

سقفی از تعداد گره‌های امیدبخش را بدست آوریم.

از این واقعیت می‌توانیم استفاده کنیم که هیچ دو وزیری در یک ستون نمی‌توانند قرار داشته باشند.

پس سقف گره‌های امیدبخش برابر است با:

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

تحلیل پیچیدگی زمانی الگوریتم‌های عقبگرد

این رویکرد هم نمی‌تواند مناسب باشد. چرا؟

اول ...

در امید بخش بودن گره‌ها کنترل عدم تقاطع قطری درنظر گرفته نمی‌شود.

دوم ...

چنانچه بخواهیم تحلیل درستی از پیچیدگی محاسباتی ارائه دهیم، باید ...

هم تعداد گره‌های امیدبخش و هم تعداد گره‌های ناامیدبخش در محاسبات آورده شود.

همچنانکه در مساله امکان دارد تعداد گره‌های ناامیدبخش بیشتر از تعداد گره‌های امیدبخش باشد.

تحلیل پیچیدگی زمانی الگوریتم‌های عقبگرد

رویکرد سوم)

الگوریتم را به صورت برنامه‌ای بنویسیم و آن را در کامپیوتری اجرا کنیم.

سپس بیاییم تعداد کل گره‌هایی که چک می‌شود را بشماریم.

راهبرد عقبگرد- الگوریتم مونت کارلو

همانگونه که قبلا اشاره شد

درخت فضای حالت هر الگوریتم شامل تعداد نمایی از گره‌ها است.

چنانچه مساله را برای دو نمونه از آن مساله با n یکسان حل کنیم امکان دارد...

یکی از آن مسائل نیاز به چک کردن تعداد کمی از گره‌های درخت فضای حالت داشته باشد و ...

نمونه دیگر از مساله شاید نیاز به چک کردن تمام گره‌های درخت فضای حالت را داشته باشد.

پس با این اوصاف نمی‌توانیم شبیه الگوریتم‌های قبلی تابع دقیقی برای پیچیدگی محاسباتی (کارایی) ارائه دهیم.

الگوریتم مونت‌کارلو، تخمینی از کارایی یک الگوریتم عقبگرد ارائه می‌دهد.

راهبرد عقبگرد- الگوریتم مونت کارلو

در الگوریتم مونت‌کارلو هم مانند رویکرد سوم، برنامه‌ای کامپیوتری نوشته می‌شود که در آن ...

تعداد گره‌های مشاهده شده شمرده می‌شود ولی ...

تفاوت آن با رویکرد سوم در آن است که نمی‌آید کل مسیرهای عمقی در درخت را طی کند

بلکه ...

به جای همه مسیرها، یک مسیر نمونه را طی می‌کند و سپس

براساس همان مسیر نمونه که طی کرده‌است، تخمینی از تعداد گره‌هایی که به ازای آن نمونه از مساله مشاهده می‌شود را ارائه می‌دهد.

یعنی تخمینی از تعداد گره‌ها در درخت فضای حالت هرس شده ارائه می‌دهد.

راهبرد عقبگرد- الگوریتم مونت کارلو

برای اینکه بتوان از الگوریتم مونت‌کارلو استفاده کرد باید دو شرط زیر برقرار باشد:

الف) در هر سطحی از درخت فضای حالت، بایستی همه گره‌ها تعداد یکسانی از فرزند داشته باشند.

ب) در هر سطح از درخت فضای حالت، بایستی تابع امیدبخش یکسانی برای تمامی گره‌ها وجود داشته باشد.

راهبرد عقبگرد- الگوریتم مونت کارلو

روش مونت‌کارلو به شرح زیر است:

فرض کنید m_0 تعداد فرزندان امیدبخش ریشه باشد.

یکی از این فرزندان امیدبخش ریشه را به صورت تصادفی انتخای می‌کنیم و فرض کنیم m_1 تعداد فرزندان امیدبخش آن باشد.

یکی از فرزندان امیدبخش آن را به صورت تصادفی انتخاب می‌کنیم و فرض کنیم m_2 تعداد فرزندان امیدبخش آن باشد.

...

یکی از فرزندان امیدبخش مرحله قبل را به صورت تصادفی انتخاب می‌کنیم و فرض می‌کنیم m_i تعداد فرزندان امیدبخش آن باشد.

....

راهبرد عقبگرد- الگوریتم مونت کارلو

این فرایند تازمانی ادامه می‌یابد که ...

گره انتخابی هیچ فرزند امیدبخشی نداشته باشد

یا گره انتخابی یکی از برگهای درخت فضای حالت باشد.

به دلیل آنکه فرض کردیم که:

گره‌هایی که در سطح یکسانی هستند همگی تعداد یکسانی فرزند دارند و ...

m_i تخمینی از میانگین فرزندان امیدبخش هر گره در سطح i باشد ...

همچنین با فرض اینکه t_i تعداد فرزندان هر گره در سطح iام باشد ...

بنابراین تخمین تعداد گره‌هایی که در الگوریتم عقبگرد مشاهده می‌شوند به صورت زیر درنظر گرفته می‌شود:

$$1 + t_0 + m_0 t_1 + m_0 m_1 t_2 + \cdots + m_0 m_1 \cdots m_{i-1} t_i + \cdots$$

$$1 + t_0 + m_0 t_1 + m_0 m_1 t_2 + \cdots + m_0 m_1 \cdots m_{i-1} t_i + \cdots$$

```
function numnodes=estimate()
    v=root of state space tree;
    numnodes=1;
    m=1;
    mprod=1;
    while(m~=0)
        t=number of children of v;
        mprod=mprod*m;
        numnodes=numnodes+mprod*t;
        m=number of promising children of v;
        if(m~=0)
            v=randomly selected promising child of v;
        end
    end
end
```

```
function numnodes=estimate_n_queens(n)
   numnodes=1;
   m=1;
   mprod=1;
   i=0;
   while(m~=0 && i~=n)
      mprod=mprod*m;
      numnodes=numnodes+mprod*n;
      i=i+1;
      m=0;
      promchildren=O;
      for j=1:n
         col(i)=j;
         if promising(i)
            m=m+1;
            promchildren=promchildren Union {j};
         end
      end
      if (m~=0)
         j=random selection from promchildren;
         col(i)=j;
      end
   end
end
```

راهبرد عقبگرد- الگوریتم مونت کارلو

در الگوریتم مونت‌کارلو هم مانند رویکرد سوم، برنامه‌ای کامپیوتری نوشته می‌شود که در آن ...

تعداد گره‌های مشاهده شده شمرده می‌شود ولی ...

تفاوت آن با رویکرد سوم در آن است که نمی‌آید کل مسیرهای عمقی در درخت را طی کند

بلکه ...

به جای همه مسیرها، یک مسیر نمونه را طی می‌کند و سپس

براساس همان مسیر نمونه که طی کرده‌است، تخمینی از تعداد گره‌هایی که به ازای آن نمونه از مساله مشاهده می‌شود را ارائه می‌دهد.

یعنی تخمینی از تعداد گره‌ها در درخت فضای حالت هرس شده ارائه می‌دهد.

الگوریتم عقبگرد برای مسئله کوله‌پشتی صفر و یک

$$S = \{item_1, item_2, ..., item_n\}$$

$w_i =$ weight of $item_i$

$p_i =$ profit of $item_i$

$W =$ maximum weight the knapsack can hold

که w_i و p_i و W همگی اعداد مثبتی هستند. می‌خواهیم زیر مجموعه A از S را به گونه‌ای مشخص کنیم که ...

$$\sum_{item_i \in A} p_i \quad \text{is maximized subject to} \quad \sum_{item_i \in A} w_i \leq W$$

الگوریتم عقبگرد برای مسئله کوله‌پشتی صفر و یک

این مساله‌ای بهینه‌سازی است.

این مساله به صورت چهارتایی (I, f, m, g) بیان می‌شوند که ....

I مجموعه نمونه‌ها است. در اینجا ...

تمای آیتم‌هایی که می‌توانند بررسی شوند.

xای که زیرمجموعه‌ای از I است را درنظر بگیریم. مثلا ...

مجموعه‌ای که حاوی تعدادی آیتم باشد (S)

$f(x)$ مجموعه راه حل‌های ممکن یعنی

تمامی زیرمجموعه‌های x

با فرض اینکه y، یکی از این راه حل‌ها باشد،

$m(x,y)$ بیانگر مجموع منافع (profit) در مجموعه y

g، تابع هدف است که در اینجا max می‌باشد.

الگوریتم عقبگرد برای مسئله کوله‌پشتی صفر و یک

در حل مسائل بهینه‌سازی با رویکرد عقبگرد، تازمانیکه جستجوی تمامی گره‌ها تمام نشود اطمینان نداریم که آیا گره‌ای دربرگیرنده راه‌حل است یا خیر.

بنابراین رویکرد عقبگرد با کمی تفاوت صورت می‌پذیرد.

اگر با رسیدن به گره‌ای منفعتی بیشتر از بهترین منفعت بدست آمده تاکنون باشند ...

مقدار بهترین منفعت بدست آمده تا اکنون را تغییر می‌دهیم.

الگوریتم عقبگرد برای مسئله کوله‌پشتی صفر و یک

بنابراین در مسائل بهینه‌سازی هنگامی که به یک گره می‌رسیم ابتدا منفعتی که به دست می‌آورد را محاسبه می‌کنیم و ...

چنانچه آن گره امید بخش باشد یعنی ...

انتظار برود که با گسترش فرزندان آن بهترین منفعت افزایش یابد ...

گسترش را روی فرزندانش انجام می‌دهیم.

و همین رویکرد برای بقیه فرزندان ادامه می‌یابد.

الگوریتم عقبگرد برای مسئله کوله‌پشتی صفر و یک

بنابراین در مسائل بهینه‌سازی، یک گره در صورتی امیدبخش است که ...

منفعتی بیشتر با گسترش درخت به سمت فرزندانش بتوانیم به دست آوریم.

پس هر وقت به گره‌ای وارد شدیم مسلما احتمال این بوده است که منفعتی بیشتر با مشاهده آن گره بدست آید.

پس همواره با مشاهده یک گره قبل از اینکه درباره امیدبخش بودن یا نبودنش تصمیم‌گیری کنیم باید مقدار منفعتش را بدست آوریم و با بیشتر منفعت کسب‌شده تاکنون مقایسه کنیم.

الگوریتم عقبگرد برای مسئله کوله‌پشتی صفر و یک

الگوریتم عمومی عقبگرد برای حالت مسائل بهینه‌سازی به صورت زیر است:

```
function checknode(v)
    if (value(v) is better than best)
        best=value(v);
    if promising(v)
        for each children u of v
            checknode(u);
        end
    end
end
```

الگوریتم عقبگرد برای مسئله کوله‌پشتی صفر و یک

روش ساخت درخت فضای حالت؟

به دنبال علائمی هستیم که غیر امیدبخش بودن گره را مشخص می‌کنند:

الف) در کوله‌پشتی ظرفیتی برای آیتم‌های جدید باقی نمانده باشد.

بنابراین چنانچه weight مجموع وزن آیتم‌هایی باشد که تا آن زمان اضافه شده‌اند ...

گره غیرامیدبخش است چنانچه

$$weight \geq W.$$

الگوریتم عقبگرد برای مسئله کوله‌پشتی صفر و یک

در اینجا چنانچه weight برابر با W هم باشد باز هم گره امیدبخش نیست

چرا که ...

در مسائل بهینه‌سازی گره‌ای امیدبخش است که ...

بتوان پاسخ را در گسترش فرزندانش جستجو کرد.

الگوریتم عقبگرد برای مسئله کوله‌پشتی صفر و یک

ب) به دنبال علامتی هستیم که در نگاه اول مشهود نیست ولی با توضیحات درباره آن می‌تواند مشخص گردد.

مسلما باید آیتم‌هایی را ابتدا برداریم که ارزش واحد وزنشان بیشتر باشد. پس در ایجاد درخت فضای حالت ...

ابتدا تمامی آیتم‌ها را بر اساس مقدار p_i / w_i نزولی مرتب می‌کنیم.

الگوریتم عقبگرد برای مسئله کوله‌پشتی صفر و یک

فرض کنید می‌خواهیم تصمیم‌گیری کنیم که آیا گره‌ای امیدبخش است یا خیر

توضیحات: غیر مساله کوله‌پشتی صفر و یک مساله کوله پشتی کسری هم مطرح می‌شود که در آن می‌توان بخشی از یک آیتم را نیز به مجموعه A اضافه کرد.

مسلما نمی‌توانیم به منفعتی بیشتر از هنگامی که ...

با روش کوله‌پشتی کسری می‌توانیم آیتم‌ها را برداریم دست پیدا کنیم.

الگوریتم عقبگرد برای مسئله کوله‌پشتی صفر و یک

بنابراین می‌توانیم حدبالایی از منفعت که بواسطه توسعه هر گره حاصل می‌شود را بدست آورد.

الگوریتم عقبگرد برای مسئله کوله‌پشتی صفر و یک

فرض کنید **profit** برابر با مجموع منفعتی‌هایی باشد که تا آن گره بدست آمده است.

همچنین **weight** مجموع وزن آیتم‌ها تا آن گره باشد.

دو متغیر **bound** و **totweight** را با profit و wight مقداردهی اولیه می‌کنیم.

الگوریتم عقبگرد برای مسئله کوله‌پشتی صفر و یک

سپس بقیه آیتم‌ها را شروع می‌کنیم به اضافه کردن به گونه‌ایکه وزن آنها به totweight و منفعت آنها به bound اضافه شود

تا کی؟

تا زمانی که اضافه کردن آن آیتم سبب شود که totweight بزرگتر از W شود.

در این صورت تنها بخشی از آن را اضافه می‌کنیم و

ارزش معادل با بخش اضافه شده را به bound اضافه می‌کنیم.

الگوریتم عقبگرد برای مسئله کوله‌پشتی صفر و یک

بنابراین bound برابر با سقف منفعتی می‌شود که با توسعه آن گره به فرزندانش می‌تواند به دست آید.

الگوریتم عقبگرد برای مسئله کوله‌پشتی صفر و یک

فرض کنید که گره در سطح i است و ...

و گره‌ای که در سطح k است سبب می‌شود که ...

مجموع وزن‌ها بیشتر از W گردد. بنابراین ...

$$totweight = weight + \sum_{j=i+1}^{k-1} w_j$$

$$bound = \underbrace{\left(profit + \sum_{j=i+1}^{k-1} p_j \right)}_{\text{Profit from first } k-1 \text{ items taken}} + \underbrace{(W - totweight)}_{\text{Capacity available for } k\text{th item}} \times \underbrace{\frac{p_k}{w_k}}_{\text{Profit per unit weight for } k\text{th item}}$$

الگوریتم عقبگرد برای مسئله کوله‌پشتی صفر و یک

$$bound = \underbrace{\left(profit + \sum_{j=i+1}^{k-1} p_j \right)}_{\text{Profit from first } k-1 \text{ items taken}} + \underbrace{(W - totweight)}_{\text{Capacity available for } k\text{th item}} \times \underbrace{\frac{p_k}{w_k}}_{\text{Profit per unit weight for } k\text{th item}}.$$

چنانچه maxprofit برابر با بهترین منفعت به دست آمده تا آن گره باشد، سپس ...

گره در سطح i امیدبخش نیست چنانچه ...

$$bound \leq maxprofit.$$

```
function knapsack(i,profit,weight)
    if(weight<=W && profit>maxProfit)
        maxProfit=profit;
        numbest=i;
        bestSet=include;
    end
    if promising(i,weight,profit)
        include(i+1)=true;
        knapsack(i+1,profit+p(i+1),weight+w(i+1));
        include(i+1)=false;
        knapsack(i+1,profit,weight);
    end
end
```

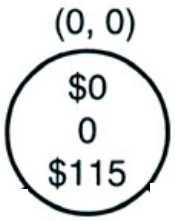
الگوریتم عقبگرد برای مسئله کوله‌پشتی صفر و یک

فرض کنید که برای $n=4$ و $W=16$ مقادیر زیر را داریم:

i	p_i	w_i	$\frac{p_i}{w_i}$
1	\$40	2	\$20
2	\$30	5	\$6
3	\$50	10	\$5
4	\$10	5	\$2

الگوریتم عقبگرد برای مسئله کوله‌پشتی صفر و یک

در این مثال، آیتم‌ها بر اساس p_i / w_i به صورت نزولی مرتب شده‌اند.



Item 1 $\begin{bmatrix} \$40 \\ 2 \end{bmatrix}$

Item 2 $\begin{bmatrix} \$30 \\ 5 \end{bmatrix}$

Item 3 $\begin{bmatrix} \$50 \\ 10 \end{bmatrix}$

Item 4 $\begin{bmatrix} \$10 \\ 5 \end{bmatrix}$

در ادامه، بازهم درباره مسائل بهینه‌سازی صحبت خواهیم کرد.
باز هم به عنوان مثال مساله کوله‌پشتی صفرویک مطرح خواهد شد.
رویکردی که در ادامه مظرح خواهد شد، ...
شاخه و حد (branch and bound) هست که ...
توسعه‌ای از رویکرد عقبگرد می‌باشد.

راهبرد شاخه و حد

در رویکرد شاخه‌وحد نیز مانند رویکرد عقبگرد از ...

درخت فضای حالت استفاده می‌کنیم.

تفاوت این دو رویکرد در این است که:

(1) در شاخه‌وحد محدود نیستیم تا برای پیمایش درخت فضای حالت فقط از پیمایش ...

Preorder استفاده کنیم. بلکه ...

می‌توانیم از هر نوع پیمایش سیستماتیک دیگر یا خلاقانه استفاده کنیم

(2) روش شاخه و حد فقط برای مسائل بهینه‌سازی مناسب است.

راهبرد شاخه و حد

در این رویکرد برای هر گره در درخت فضای حالت، حد (bound) ای محاسبه می‌شود تا
مشخص شود که آن گره امیدبخش است یا خیر.
bound هر گره بیانگر حدی از مقدارهای m(x,y) است که با گسترش آن گره به دست می‌آید.
اگر bound از بهترین m(x,y) ای که تاکنون بدست آمده‌است بهتر نباشد در این صورت ...
گره امیدبخش نیست ودرغیراینصورت
امیدبخش است.

راهبرد شاخه و حد

با این توضیحات الگوریتم عقبگرد ارائه شده برای مساله کوله‌پشتی صفرویک عملا الگوریتم ...

شاخه و حد است چراکه ...

در آن الگوریتم هم گره امیدبخش نبود چنانچه bound از maxprofitای که تا آن زمان بدست آمده بود بزرگتر نبود.

با این اوصاف وقتیکه bound مربوط به همه گره‌های امیدبخش را داریم **بهتر** است تا ...

فرزندان گره‌ای را ایجاد کنیم که ....

دارای بیشترین bound است.

با این رویکرد می‌توانیم سریعتر به راه حل بهینه دست یابیم تا اینکه همانند رویکرد عقبگرد ....

به صورت preorder پیمایش را ادامه دهیم.

این رویکرد «جستجوی اولین-بهترین با هرس‌کردن شاخه و حد» نامیده می‌شود.

راهبرد شاخه و حد

علاوه بر این رویکرد می‌توانیم رویکرد ساده‌تر «جستجوی سطح اول با هرس کردن شاخه و حد» را داشته باشیم.

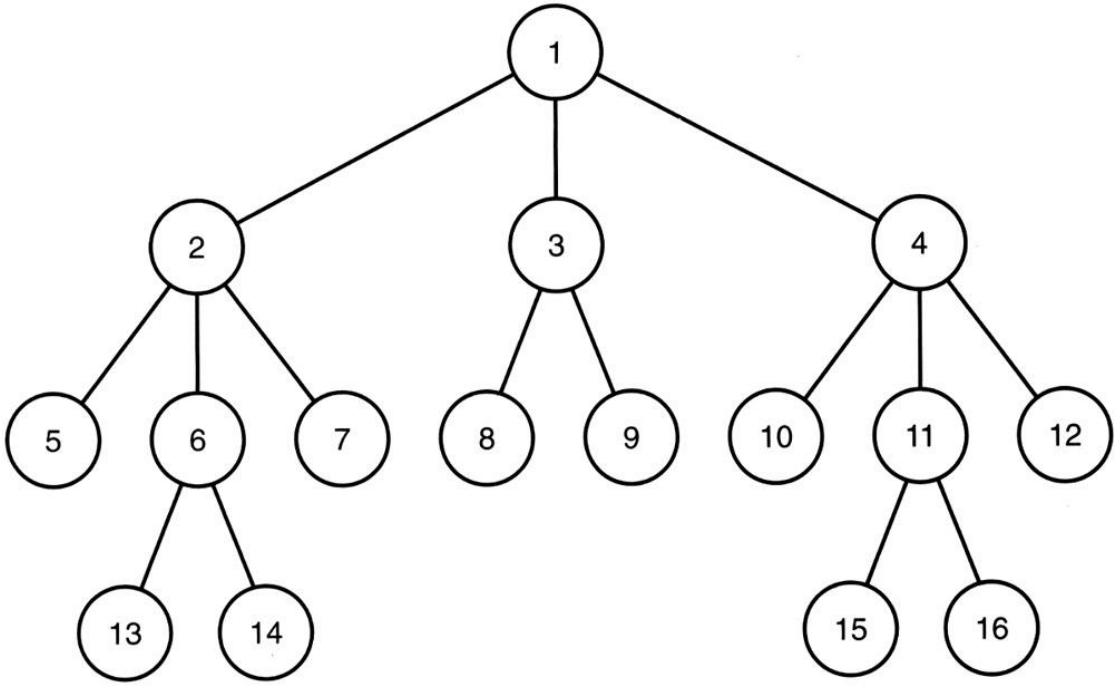
راهبرد شاخه و حد

رویکرد جستجوی سطح اول با هرس کردن شاخه و حد شامل:

1- ابتدا مشاهده ریشه

2- سپس تمامی گره‌های در سطح اول

3- سپس تمامی گره‌های در سطح دوم و ...

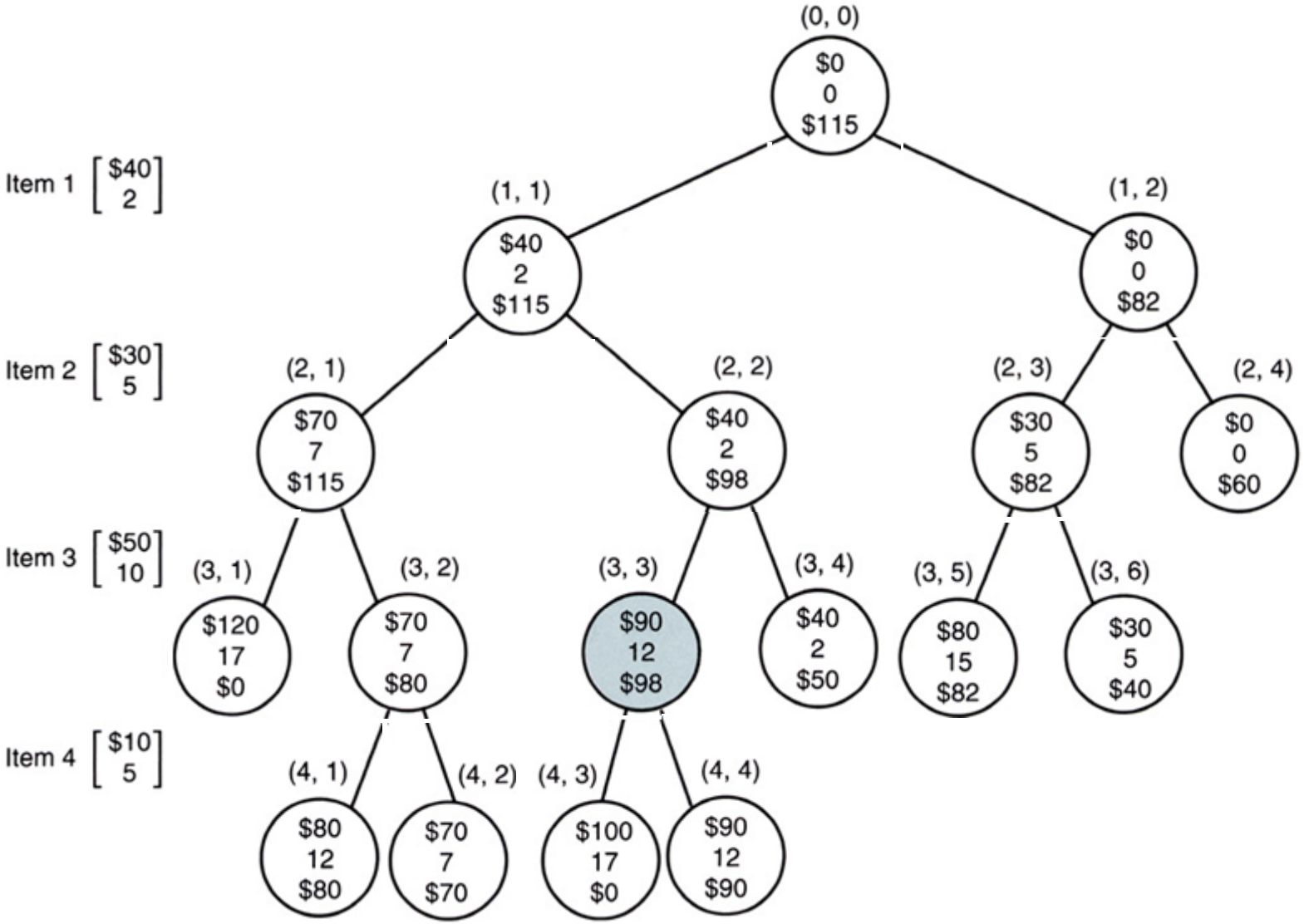


راهبرد شاخه و حد

مساله کوله‌پشتی زیر با n=4 و W=16 را درنظر بگیرید:

i	p_i	w_i	$\frac{p_i}{w_i}$
1	\$40	2	\$20
2	\$30	5	\$6
3	\$50	10	\$5
4	\$10	5	\$2

راهبرد شاخه و حد



راهبرد شاخه و حد

«جستجوی اولین-بهترین با هرس‌کردن شاخه و حد»

به صورت عمومی استراتژی جستجوی سطح اول مزیتی نسبت به رویکرد عقبگرد ندارد. چراکه ...

در آنجا درخت به صورت عمقی پیمایش می‌شد و در اینجا به صوت سطی

جستجوی سطح اول می‌تواند با پیشنهادهای زیر زودتر پاسخ بهینه را پیدا کند:

بعد از آنکه تمامی فرزندان یک گره مشاهد شد ...

به جای اینکه بیاییم و از اول صف گره بعدی را برداریم و فرزندان آن را مشاهده کنیم ...

بیاییم گره‌ی را از صف برداریم که دارای bound بزرگتری از بقیه باشد و فرزندان آن را مشاهده کنیم.

این رویکرد دارای سرعت همگرایی بیشتری نسبت به دو حالت قبل است.

راهبرد شاخه و حد