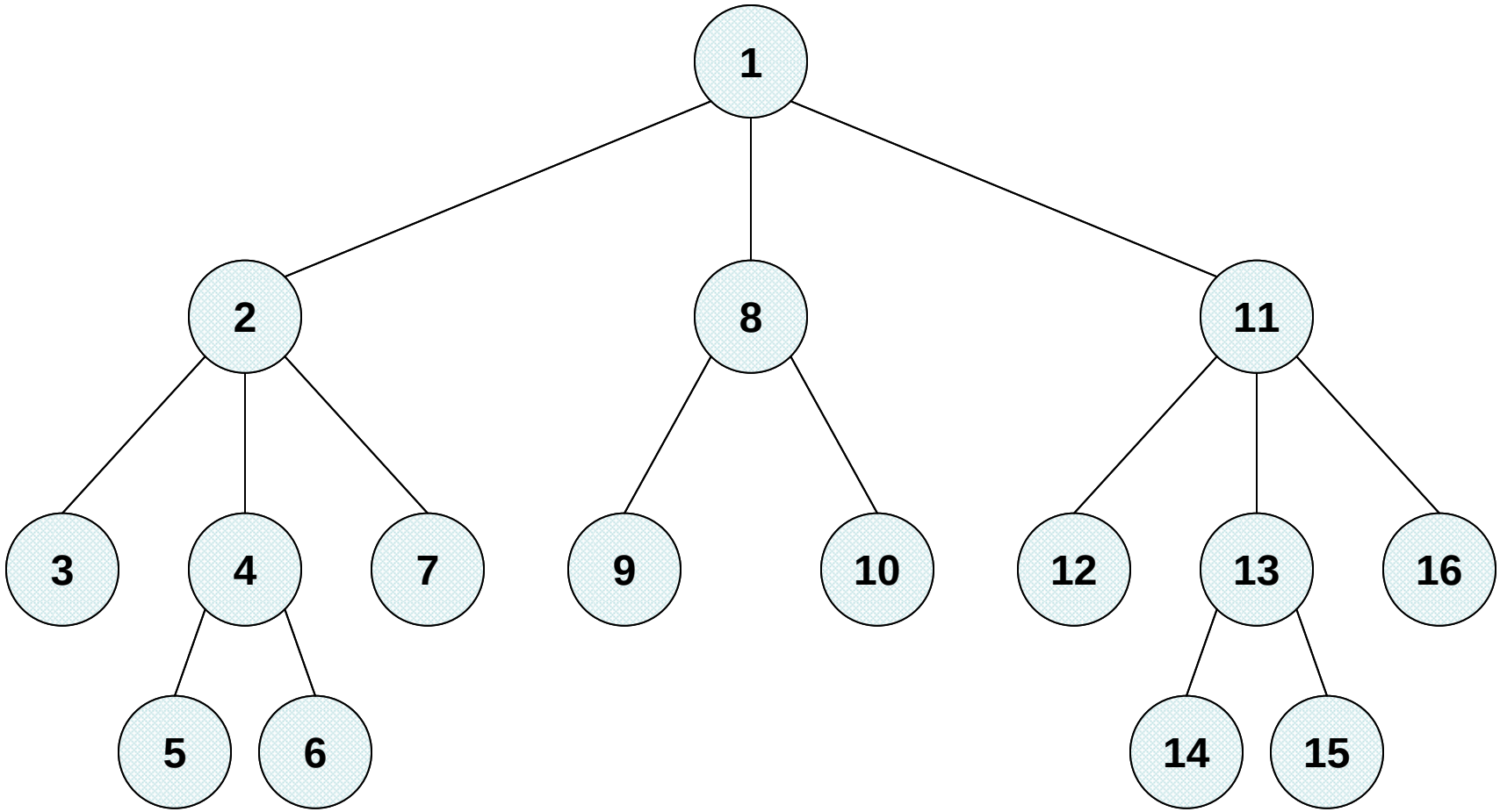
فصل پنجم
عقبگرد

ایده

- پرچین پر پیچ و خم قصر هامپتون
- انتخاب دنباله ای از اشیاء از یک مجموعه مشخص به طوری که معیار خاصی برآورده شود
- مثال: مساله n–وزیر
  - دنباله: n موقعیت روی صفحه شطرنج
  - مجموعه: n^2 موقعیت ممکن
  - معیار: هیچ دو وزیری همدیگر را تهدید نکنند.
- جستجوی اول عمق درخت (پیمایش پیش ترتیب درخت)

جستجوی اول عمق



شکل ۵-۱ یک درخت که در آن گره ها طبق جستجوی عمقی شماره گذاری شده اند.

الگوریتم جستجوی اول عمق

```
void depth_first_tree_search ( node v )
{
   node u ;
   visit v ;
   for ( each child u of v )
         depth_first_tree_search ( u ) ;
}
```

مساله ۴-وزیر

- درخت فضای حالت

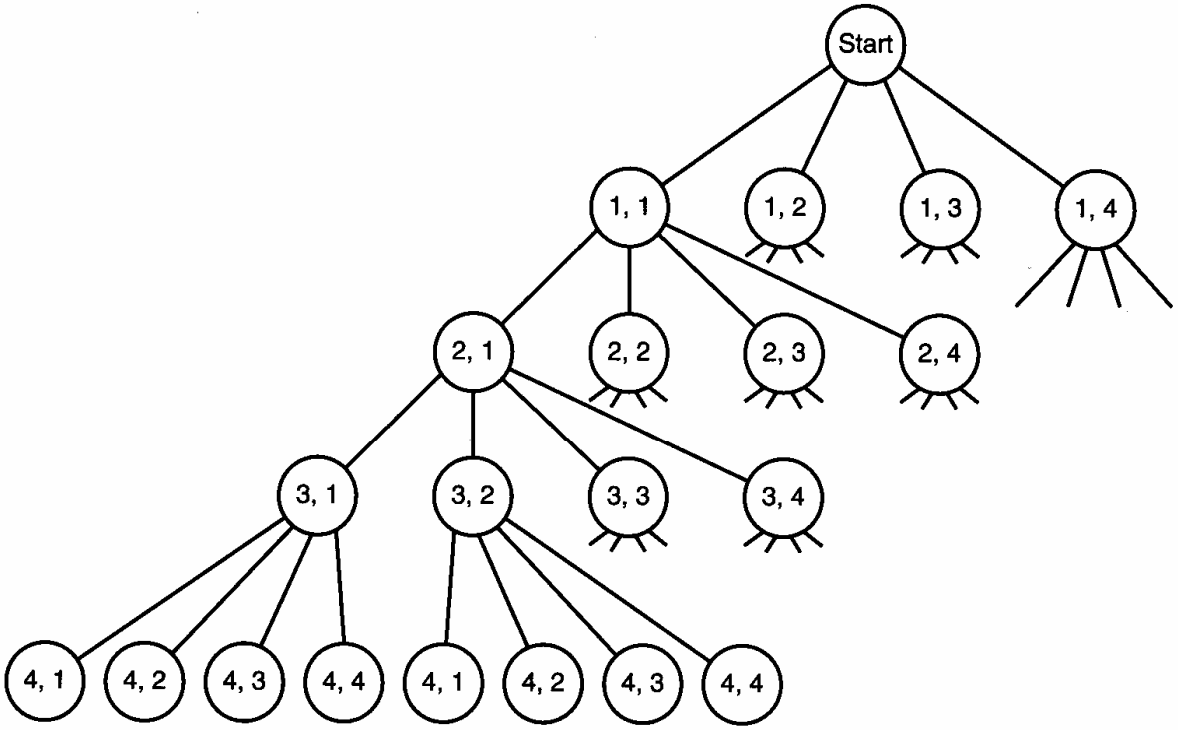


Figure 5.2 ● A portion of the state space tree for the instance of the *n*-Queens problem in which $n = 4$. The ordered pair <*i*, *j*>, at each node means that the queen in the *i* th row is placed in the *j* th column. Each path from the root to a leaf is a candidate solution.

در صورت بررسی هر راه حل کاندیدا ...

[<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>]

...

جستجو برای علامت های بن بست

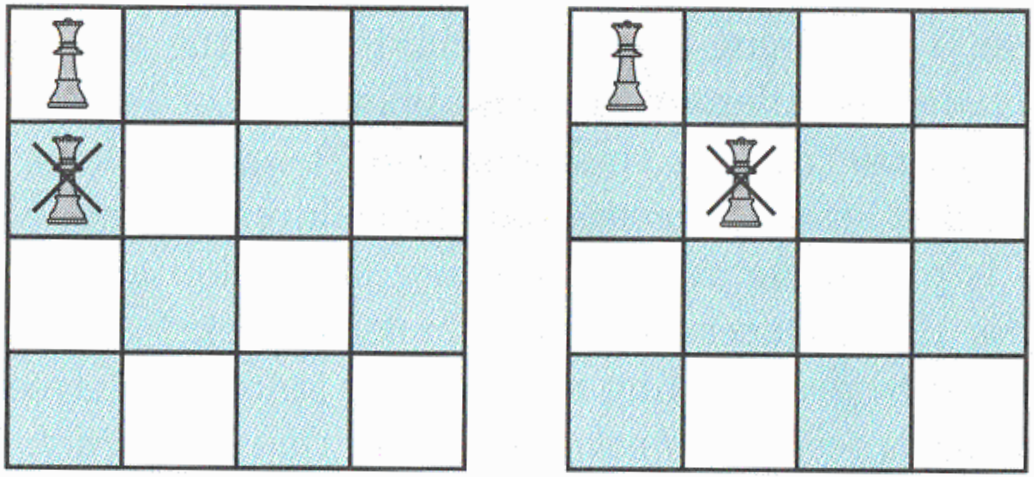


Figure 5.3 • Diagram showing that if the first queen is placed in column 1, the second queen cannot be placed in column 1 (a) or column 2 (b).

عقبگرد (*Backtracking*)

- عقبگرد: روالی که توسط آن، پس از تعیین اینکه گره ای غیر امید بخش می باشد، به گره پدر آن عقبگرد می کنیم و جستجو را در فرزند دیگری از گره پدر ادامه می دهیم.
- گره غیر امید بخش
  - گره ای که هنگام ملاقات آن دریابیم که احتمالا منجر به راه حل نمی شود
- گره امید بخش
- هرس کردن درخت فضای حالت
- درخت فضای حالت هرس شده

الگوریتم کلی

```
void checknode ( node v)
{
    node u ;

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

مساله ۴-وزیر

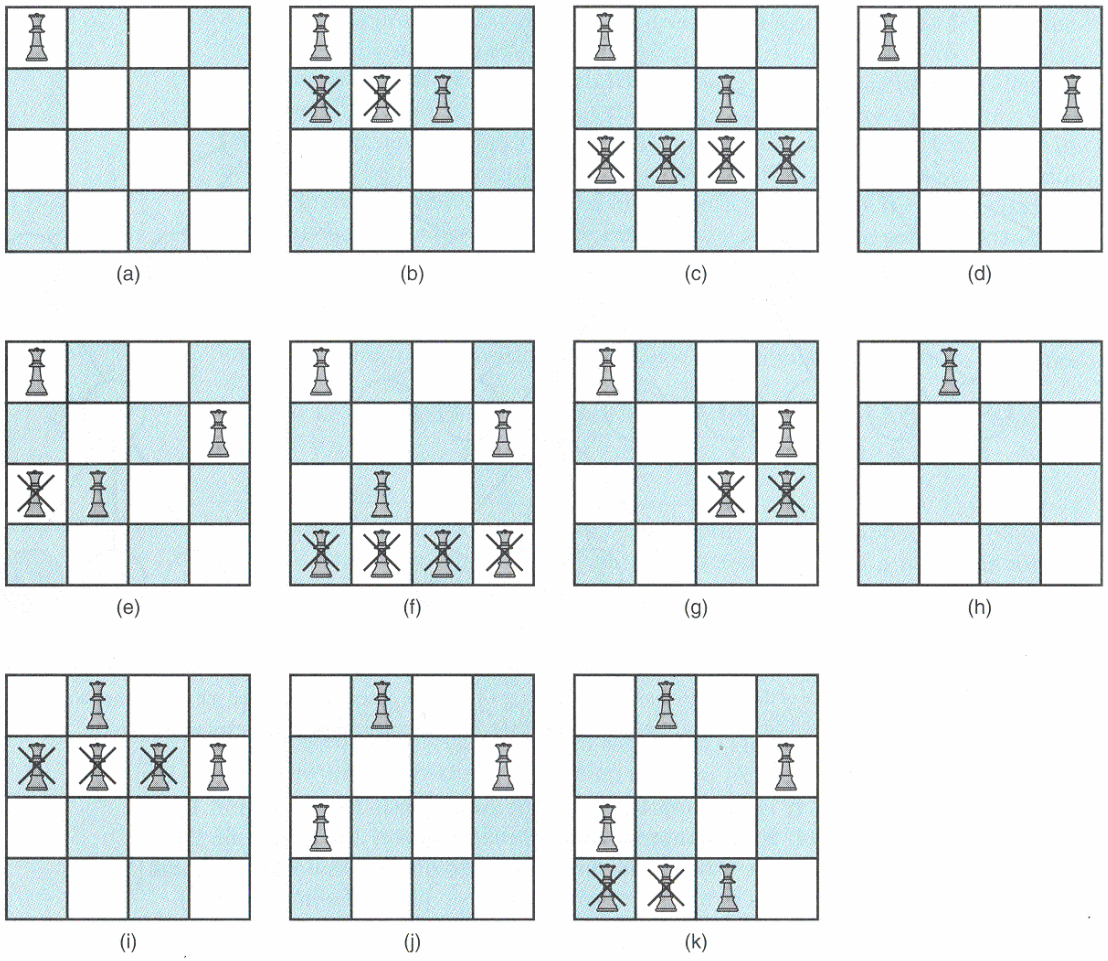
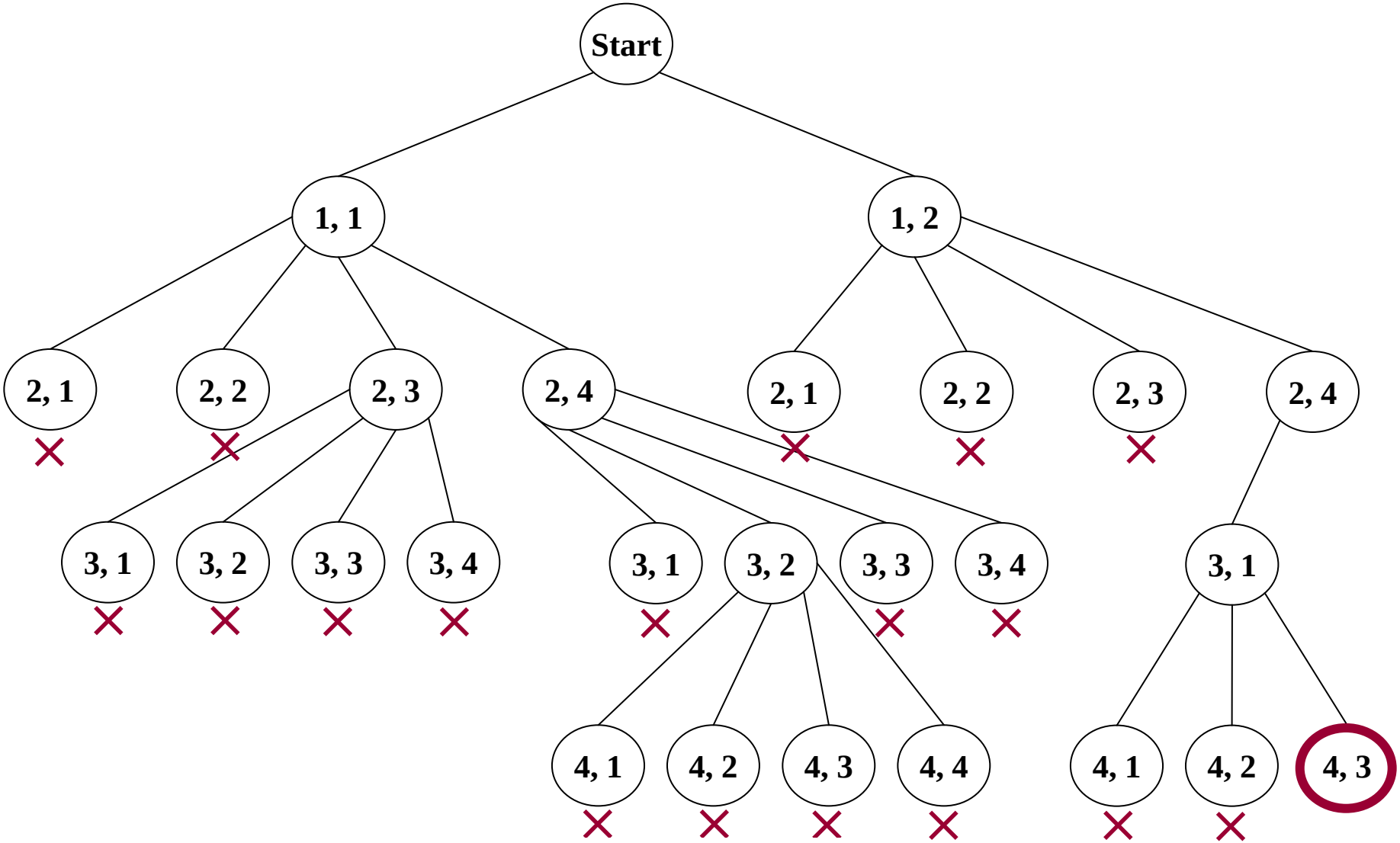


Figure 5.5 ● The actual chessboard positions that are tried when backtracking is used to solve the instance of the *n*-Queens problem in which $n = 4$. Each nonpromising position is marked with a cross.

مثال: جستجوی عقبگرد برای ۴-وزیر



اجتناب از تولید گره های غیر امید بخش

```
void expand ( node v )
{
    node u ;

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

مساله n-وزیر

- بررسی اینکه آیا دو وزیر یکدیگر را تهدید می کنند
- بررسی ستون ها

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

- بررسی قطرها

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

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

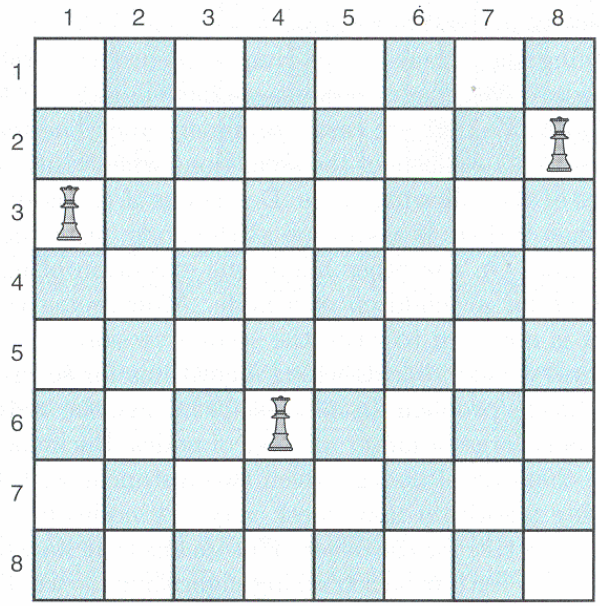


Figure 5.6 ● The queen in row 6 is being threatened in its left diagonal by the queen in row 3 and in its right diagonal by the queen in row 2.

الگوریتم

- مساله: قرار دادن n وزیر روی یک صفحه شطرنج $n \times n$ به گونه ای که هیچ دو وزیری در یک سطر، ستون و یا قطر قرار نگیرند.

- ورودی ها: عدد صحیح و مثبت n

- خروجی ها: تمامی راه های ممکن برای قرار دادن n وزیر در یک صفحه شطرنج $n \times n$ به طوری که هیچ دو وزیری نتوانند یکدیگر را تهدید کنند. هر خروجی شامل آرایه ای از اعداد صحیح به نام col است که از یک تا n اندیس گذاری شده است و در آن col[i] بیانگرستونی است که وزیر ردیف i اُم در آن قرار می گیرد.

الگوریتم

```
void queens( index i)
{
    if ( promising (i))
        if ( i == n)
            cout << col [1] through col [n] ;
        else
            for ( j =1; j <= n; j++) {
                col [i + 1] = j ;
                queens (i + 1) ;
            }
}
```

الگوریتم بررسی امید بخش بودن گره ها

```
bool promising ( index i )
{
    index k ;
    bool switch ;
    k = 1 ;
    switch = true ;
    while ( k < i && switch ) {
        if ( col [i] = col [k] || abs ( col [i] – col [k] ) = i – k )
            switch = false ;
        k++ ;
    }
}
```

کارآیی

- بررسی تمام درخت فضای حالت (تعداد گره های بررسی شده)

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

- استفاده از مزیت اینکه هیچ دو وزیر ی نمی توانند همزمان در یک سطر یا ستون قرار بگیرند

$1 + n + n(n\text{-}1) + n(n\text{-}1)(n\text{-}2) + \ldots + n!$ تعداد گره های امید بخش

مقایسه

● **Table 5.1** An illustration of how much checking is saved by backtracking in the n-Queens problem*

n	Number of Nodes Checked by Algorithm 1†	Number of Candidate Solutions Checked by Algorithm 2‡	Number of Nodes Checked by Backtracking	Number of Nodes Found Promising by Backtracking
4	341	24	61	17
8	19,173,961	40,320	15,721	2057
12	9.73×10^{12}	4.79×10^{8}	1.01×10^{7}	8.56×10^{5}
14	1.20×10^{16}	8.72×10^{10}	3.78×10^{8}	2.74×10^{7}

*Entries indicate numbers of checks required to find all solutions.

†Algorithm 1 does a depth-first search of the state space tree without backtracking.

‡Algorithm 2 generates the $n!$ candidate solutions that place each queen in a different row and column.

مساله حاصل جمع زیر مجموعه ها

- مثال ۵-۲:

فرض کنید

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

از آنجا که

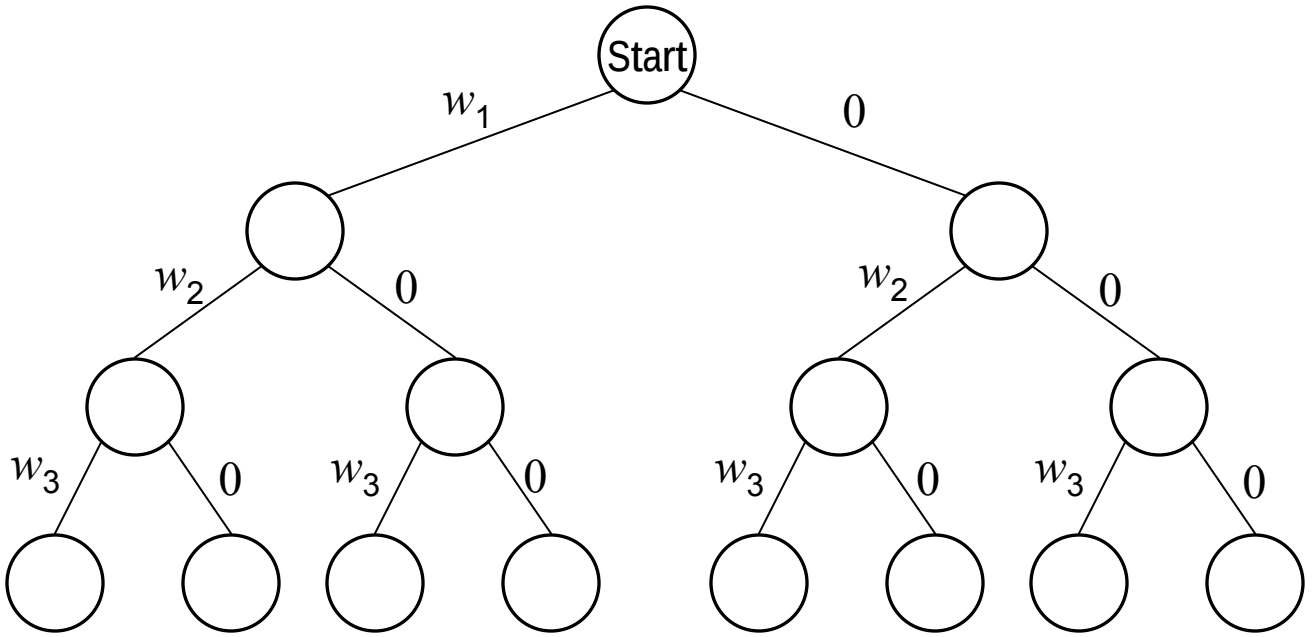
$w_1 + w_2 + w_3 = 21,$
$w_1 + w_5 = 21,$
$w_3 + w_4 = 21$

جواب ها برابرند با:

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

درخت فضای حالت

- $w_1 = 2, w_2 = 4, w_3 = 5$

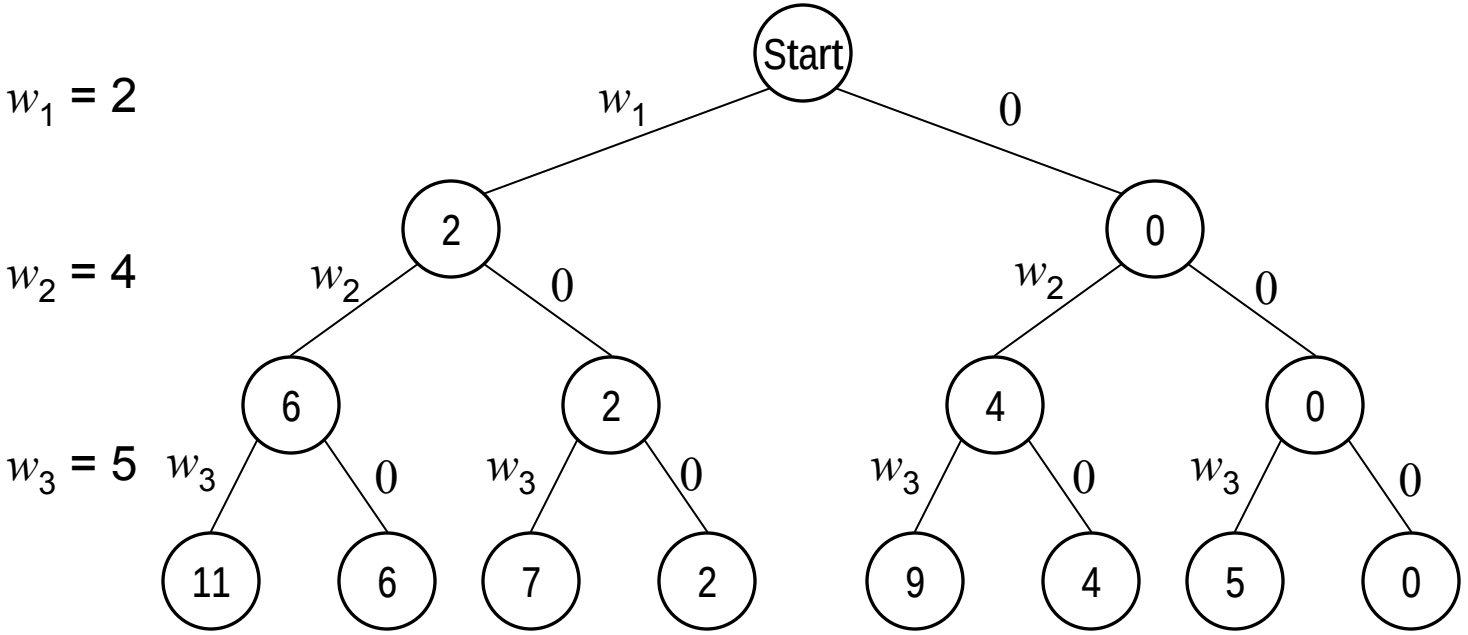


شکل ۵-۷ یک درخت فضای حالت برای نمونه ای از مساله حاصل جمع زیر مجموعه ها که در آن $n = 3$.

درخت فضای حالت

- $W = 6,$

$$w_1 = 2, w_2 = 4, w_3 = 5$$



شکل ۵-۸ یک درخت فضای حالت برای مساله حاصل جمع زیر مجموعه ها برای نمونه مثال ۵-۳. مقادیر ذخیره شده در هر گره برابر مجموع وزن ها تا آن گره می باشد.

بررسی امید بخش بودن یک گره

- مرتب سازی وزن ها به ترتیب غیر نزولی
- بررسی گره در سطح i:
  - $weight + w_{i+1} > W$
  - $weight + total < W$

درخت فضای حالت

- $W = 13$,

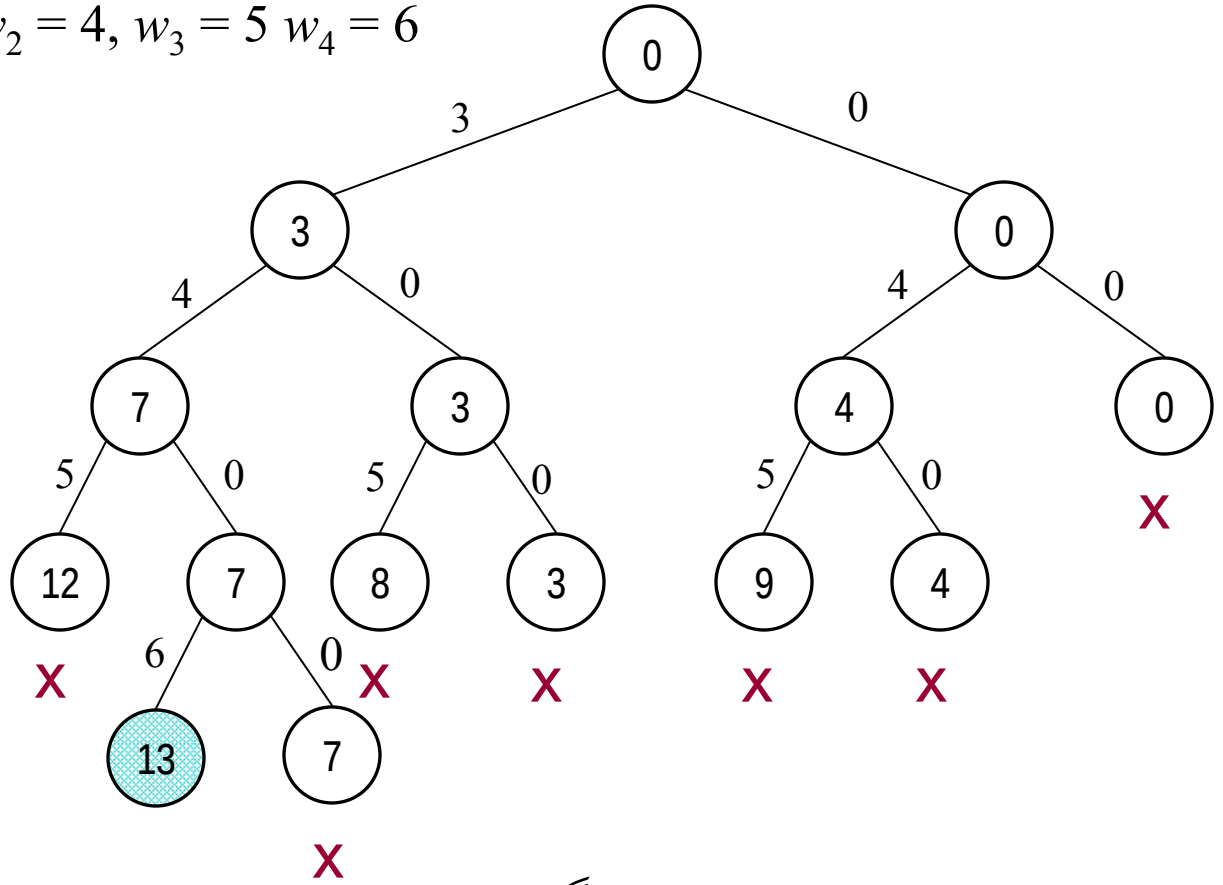
$w_1 = 3, w_2 = 4, w_3 = 5\ w_4 = 6$

$w_1 = 3$

$w_2 = 4$

$w_3 = 5$

$w_4 = 6$



شکل ۵-۹ درخت هرس شده فضای حالت توسط عقبگرد در مثال ۵-۴. مقادیر ذخیره شده در هر گره برابر مجموع وزن ها تا آن گره می باشد. تنها جواب در گره سایه خورده پیدا شده است. گره های غیر امید بخش با علامت ضرب مشخص شده اند.

الگوریتم

- مساله: تعیین همه ترکیبات اعداد صحیح موجود در یک مجموعه n عضوی، به طوری که مجموع آن ها مساوی مقدار معین W شود.

- ورودی ها: عدد صحیح مثبت n، آرایه مرتب (غیر نزولی) از اعداد صحیح مثبت که از یک تا n اندیس گذاری شده است. و عدد صحیح مثبت W.

- خروجی ها: تمام ترکیبات اعداد صحیح ورودی که مجموعشان برابر W باشد.

الگوریتم

```
void sum_of_subsets ( index i,
                      int weight, int total )
{
    if ( promising (i))
        if ( weight = W )
            cout << include [1] through include [i] ;
        else {
            include [i + 1] = "yes";
            sum_of_subsets ( i + 1, weight + w [i + 1], total - w [ i +1] ) ;
            include [i + 1] = "no" ;
            sum_of_subsets ( i + 1, weight, total – w [ i +1] ) ;
        }
}
```

الگوریتم بررسی امید بخش بودن یک گره

```
bool promising ( index i )
{
   return ( weight + total >=W) &&
          ( weight == W || weight + w [ i + 1] <= W ) ;

}
```

پیچیدگی زمانی

- اولین فراخوانی تابع

$sum\_of\_subset(0, 0, total)$

که

$$total = \sum_{j=1}^{n} w[j]$$

- تعداد گره های بررسی شده

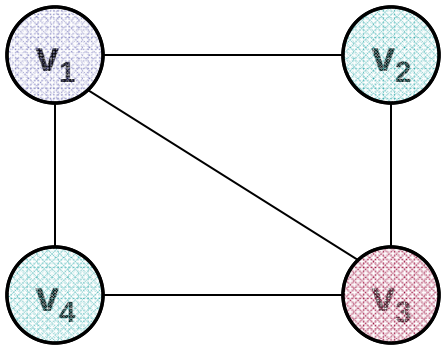
$$1 + 2 + 2^2 + \ldots + 2^n = 2^{n+1} - 1$$

رنگ آمیزی گراف

- مساله m-Coloring:

– یافتن همه راه های ممکن برای رنگ آمیزی رئوس یک گراف بدون جهت با استفاده از حداکثر m رنگ متفاوت به طوری که هیچ دو راس مجاوری هم رنگ نباشند.

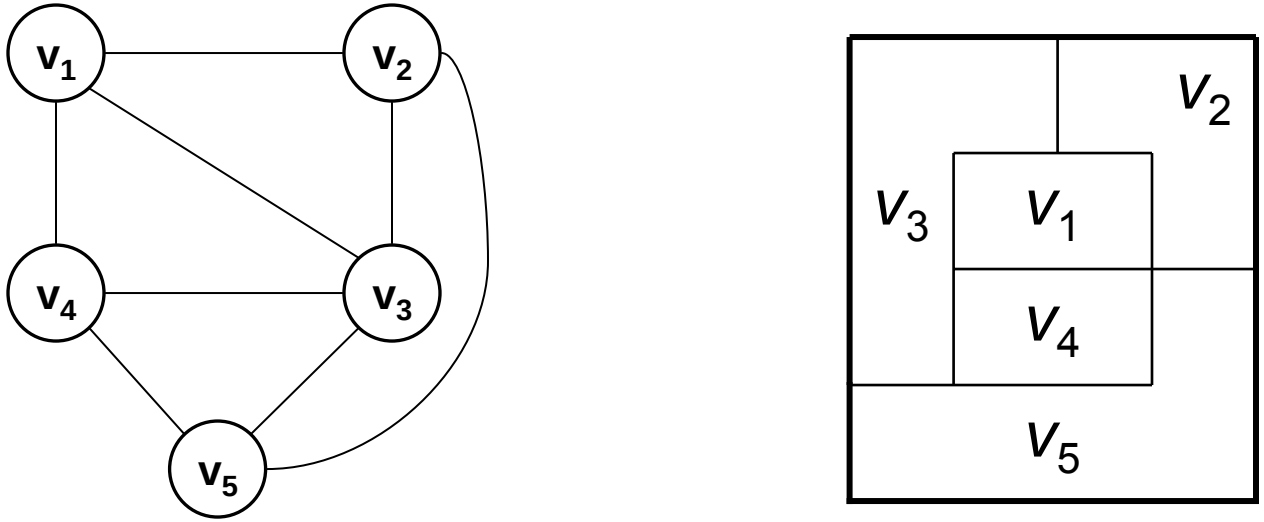
- مثال ۵-۵:



شکل ۵-۱۰ گرافی که با دورنگ قابل رنگ آمیزی نمی باشد. حل مساله رنگ آمیزی با ۳ رنگ

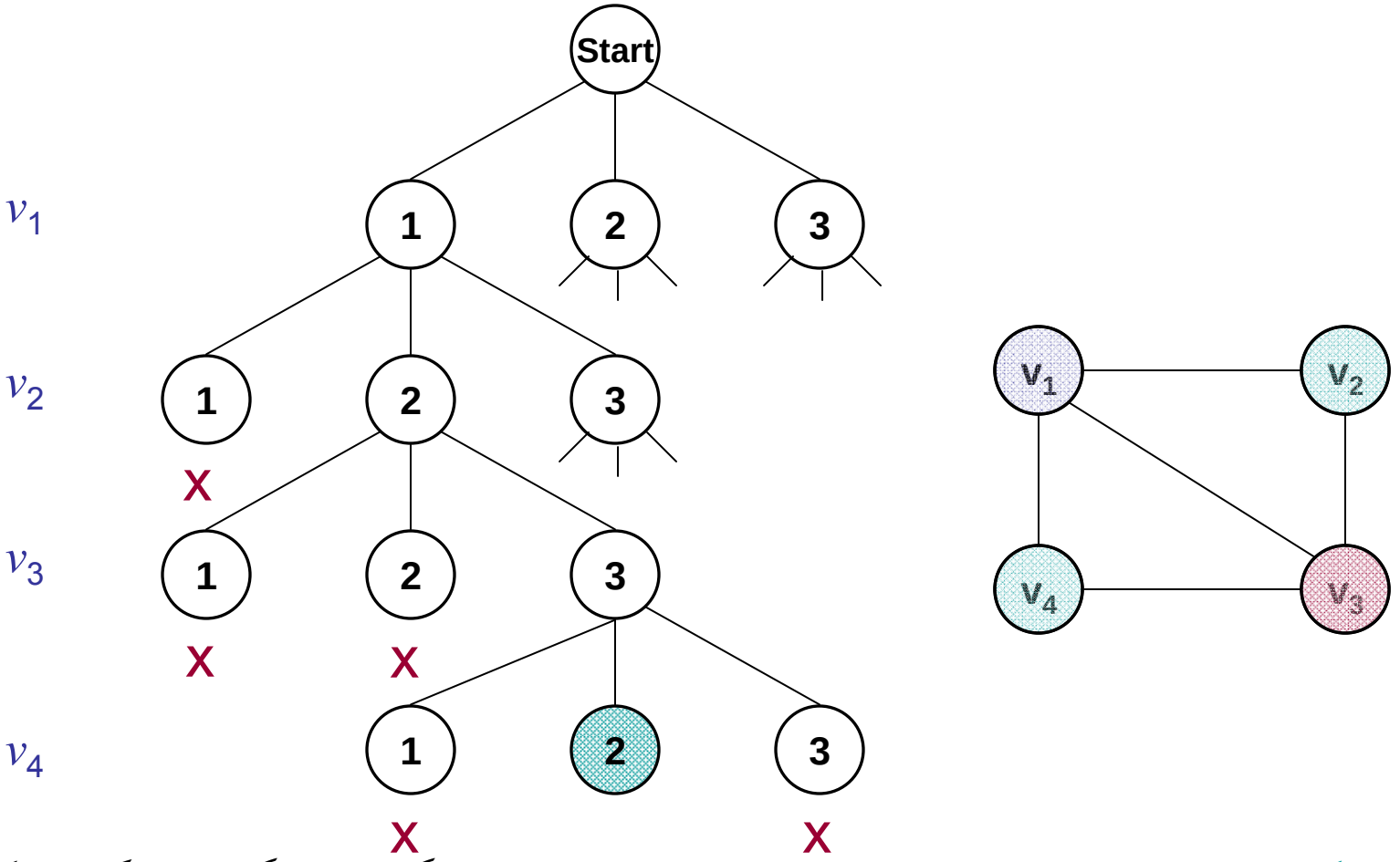
کاربرد: رنگ آمیزی نقشه

- گراف مسطح: گرافی که بتوان آن را در صفحه رسم کرد به طوری که هیچ دو یالی یکدیگر را قطع نکنند.



شکل ۵-۱۱ یک نقشه (سمت راست) و نمایش گراف مسطح مربوط به آن (سمت چپ)

رنگ آمیزی گراف با ۳ رنگ



شکل ۵-۱۲ بخشی از درخت هرس شده فضای حالت که توسط عقبگرد برای رنگ آمیزی گراف شکل ۵-۱۰ با ۳ رنگ تولید شده است. اولین راه حل در گره سایه خورده یافت شده است. گره های غیر امید بخش با علامت ضرب نشان داده شده اند.

الگوریتم

مساله: تعیین همه راه هایی که در آن رئوس یک گراف بدون جهت را می توان با حد اکثر m رنگ به گونه ای رنگ آمیزی نمود که هیچ دو راس مجاوری هم رنگ نباشند.

ورودی ها: اعداد صحیح و مثبت n و m و گراف بدون جهت حاوی n راس. گراف توسط ماتریس مجاورتی W نشان داده می شود.

خروجی ها: همه رنگ آمیزی های ممکن برای گراف ورودی با استفاده از حداکثر m رنگ به طوری که هیچ دو راس مجاوری هم رنگ نباشند. خروجی مربوط به هر رنگ آمیزی یک آرایه به نام $vcolor$ می باشند که از یک تا n اندیس گذاری شده است و در آن $vcolor[i]$ بیانگر رنگ مربوط به راس i می باشد.

الگوریتم

```
void m_coloring ( index i )
{
    int color ;
    if ( promising ( i ))
        if ( i == n)
            cout << vcolor[1] through vcolor[n] ;
        else
            for ( color = 1; color  <= m; color++) {
                vcolor[i + 1] = color ;
                m_coloring ( i + 1) ;
            }
}
```

الگوریتم بررسی امید بخش بودن یک گره

```
bool promising ( index i )
{
    index j ;
    bool switch ;
    switch = true ;
    j = 1;
    while ( j < i && switch) {
          if ( W [i] [j] && vcolor [i] == vcolor [j] )
                switch = false ;
          j++ ;
    }
    return switch ;
}
```

پیچیدگی زمانی

- فراخوانی تابع در بالاترین سطح

$m\_coloring(0)$

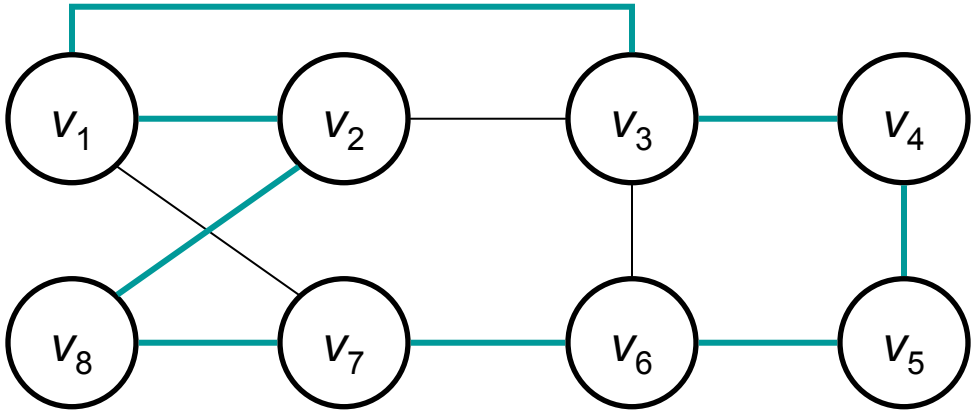
- تعداد گره ها در درخت فضای حالت

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

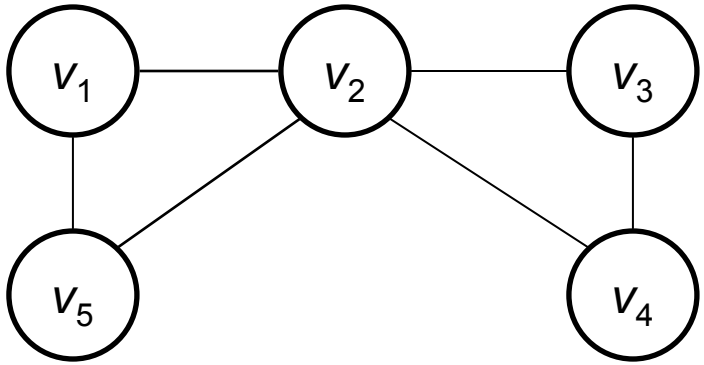
مساله دور های هامیلتونی

- یاد آوری:
  - مساله تعیین کوتاهترین تور با $n = 20$
    - روش برنامه ریزی پویا $T(n) = (n - 1)(n - 2)2^{n-3}$ (۴۵ ثانیه)
    - روش کورکورانه $(n -1)!$ (۳۸۰۰ سال)
- اگر $n = 40$
  - روش برنامه ریزی پویا ۶/۴۶ سال زمان برای تعیین کوتاهترین تور
  - مساله یافتن هر توری در گراف (بدون توجه به وزن تور) را مساله دورهای هامیلتونی می نامیم.

دور هامیلتونی



(الف)



(ب)

شکل ۵-۱۳ گراف (الف) دارای دور هامیلتونی می باشد. گراف (ب) فاقد دور هامیلتونی است.

الگوریتم عقبگرد برای مساله دورهای هامیلتونی

- مساله: تعیین کلیه دورهای هامیلتونی در یک گراف همبند و بدون جهت
- ورودی ها: عدد صحیح و مثبت n و گراف بدون جهت دارای n راس
- خروجی ها: تمام مسیرهای بسته که از یک راس مفروض آغاز شده، کلیه رئوس گراف را دقیقا یکبار ملاقات می کند و به راس شروع ختم می شود. خروجی هر مسیر، آرایه ای از اندیس های $vindex$ است که از صفر تا $n-1$ اندیس گذاری شده اند و در آن $vindex[i]$ اندیس راس i اُم روی مسیر است. اندیس راش شروع، $vindex[0]$ است.

الگوریتم

```
void hamilton ( index i )
{
    index j ;

    if ( promising (i))
        if ( i == n -1 )
                cout << vindex [0] through vindex [n - 1]  ;
        else
                for ( j = 2; j <= n; j++ ){
                        vindex[ i + 1] = j ;
                        hamilton (i + 1) ;
                }
}
```

پیچیدگی زمانی

- فراخوانی تابع در بالا ترین سطح

vindex [0] = 1;

hamilton (0) ;

- تعداد گره های درخت فضای حالت

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

الگوریتم

```
bool promising ( index i )
{
    index j ;
    bool switch ;
    if ( i == n -1 && !W [vindex [n – 1]] [vindex [0]] )
            switch = false ;
    else if ( i > 0 && !W [vindex [i – 1]] [vindex [i]] )
            switch = false ;
    else {
            switch = true ;
            j = 1 ;
            while ( j < i && switch ) {
                    if ( vindex [i] = vindex [j] )
                            switch = false;
                    j++ ;
            }
    }
    return switch ;
}
```

مساله کوله پشتی ۰-۱

- تفاوت با مسایل قبلی: بهینه سازی

– تا زمانی که جستجو به پایان نرسد نمی توان دریافت که آیا گره ای حاوی یک راه حل می باشد یا خیر.

– در حل کردن مسایل بهینه سازی توسط عقبگرد، همواره فرزندان یک گره امید بخش را ملاقات می کنیم.

الگوریتم کلی

```
void checknode ( node v )
{
   node u ;

   if ( value (v)  is better than best )
         best = value (v) ;
   if ( promising (v))
         for ( each child u of v )
               checknode (u) ;
}
```

کوله پشتی

- گره غیر امید بخش

$weight \geq W$

- مرتب سازی قطعات برحسب p_i / w_i به صورت غیر نزولی
- تعیین حد بالا برای سود قابل حصول از گسترش دادن گره
  - *profit* حاصل جمع ارزش قطعاتی که تا آن گره در نظر گرفته شده اند
  - *weight* حاصل جمع اوزان آن قطعات
  - مقدار اولیه متغیر *bound* را برابر *profit* قرار می دهیم
  - مقدار اولیه متغیر *totweight* را برابر *weight* قرار می دهیم
  - هر بار به روش حریصانه یک قطعه برداشته و ارزش آن را به *bound* و وزنش را به *totweight* اضافه می کنیم تا به قطعه ای برسیم که در صورت برداشتن آن *totweight* از W بیشتر شود. در این صورت کسری از آن را با توجه به ظرفیت باقیمانده برداشته و ارزش آن کسر را به bound اضافه می کنیم.

کوله پشتی

- فرض کنید گره در سطح i باشد و گره واقع در سطح k، گره ای باشد که حاصل جمع اوزان را از W بیشتر کند. در این صورت:

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

$$bound = \underbrace{(profit + \sum_{j=i+1}^{k-1} p_j)}_{\text{بهره حاصل از } k\text{-1 قطعه نخست}} + \underbrace{(W - totweight)}_{\text{ظرفیت باقیمانده برای قطعه } k\text{ اُم}} * \underbrace{\frac{p_k}{w_k}}_{\text{بهره واحد وزن برای قطعه } k\text{ اُم}}$$

- گره غیر امید بخش

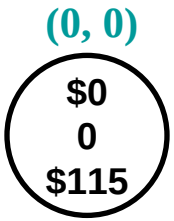
$$bound \leq maxprofit$$

مثال ٥-٦

- $n = 4, \ W = 16$

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

مثال



$$maxprofit = 0$$

$$profit = 0$$

$$weight = 0$$

$$totoweight = weight + \sum_{j=0+1}^{3-1} w_j = 0 + 2 + 5 = 7$$

$$bound = (profit + \sum_{j=0+1}^{3-1} p_j) + (W - totweight) * \frac{p_3}{w_3}$$

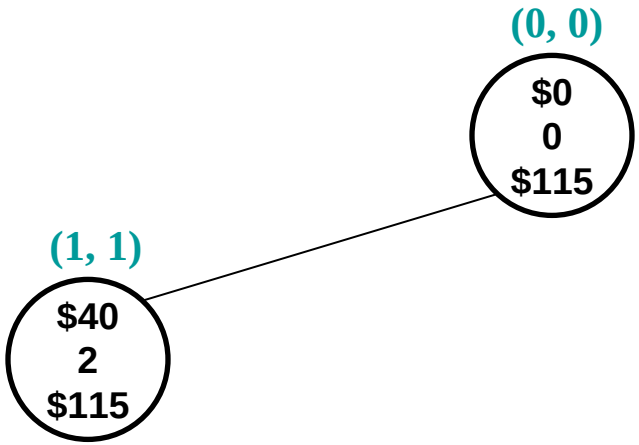
$$= \$0 + \$40 + \$30 + (16 - 7) * \frac{\$50}{10} = \$115$$

این گره امید بخش می باشد زیرا وزنش کمتر از ۱۶ (W) و حدش بزرگتر از $maxprofit = 0$ می باشد.

مثال

Item 1

($40, 2)



$$profit = \$0 + \$40 = \$40$$

$$weight = 0 + 2 = 2$$

$$maxprofit = \$40$$

$$totoweight = weight + \sum_{j=1+1}^{3-1} w_j = 0 + 2 + 5 = 7$$

$$bound = (profit + \sum_{j=1+1}^{3-1} p_j) + (W - totweight) * \frac{p_3}{w_3}$$

$$= \$0 + \$40 + \$30 + (16 - 7) * \frac{\$50}{10} = \$115$$

این گره امید بخش می باشد زیرا وزنش کمتر از ۱۶ (W) و حدش بزرگتر از $maxprofit = 40$ می باشد.

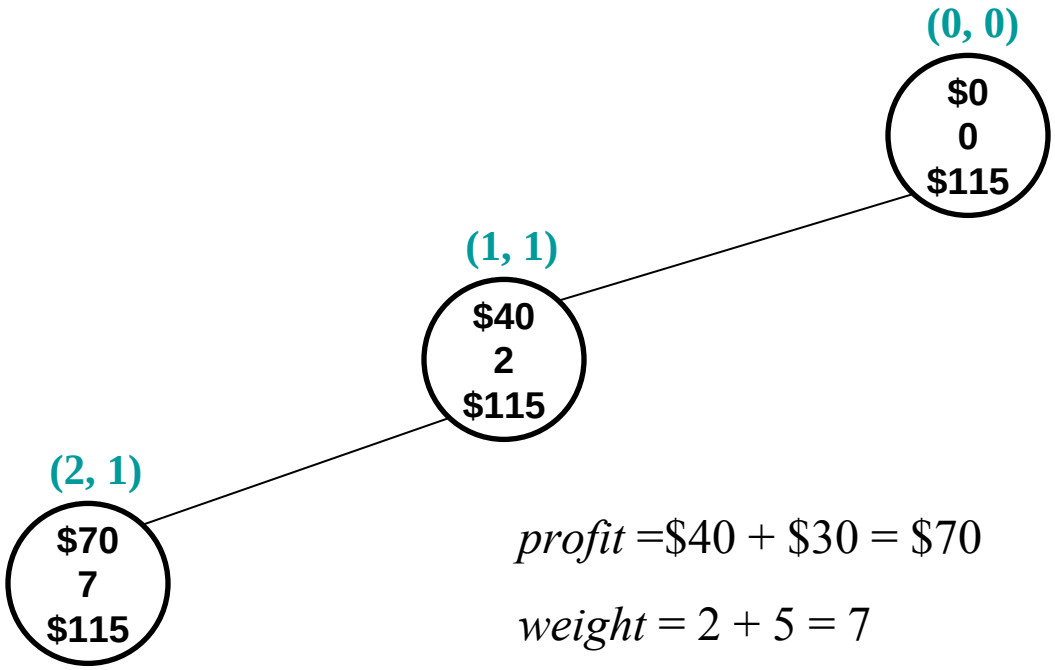
مثال

Item 1

($40, 2)

Item 2

($30, 5)



$$profit = \$40 + \$30 = \$70$$

$$weight = 2 + 5 = 7$$

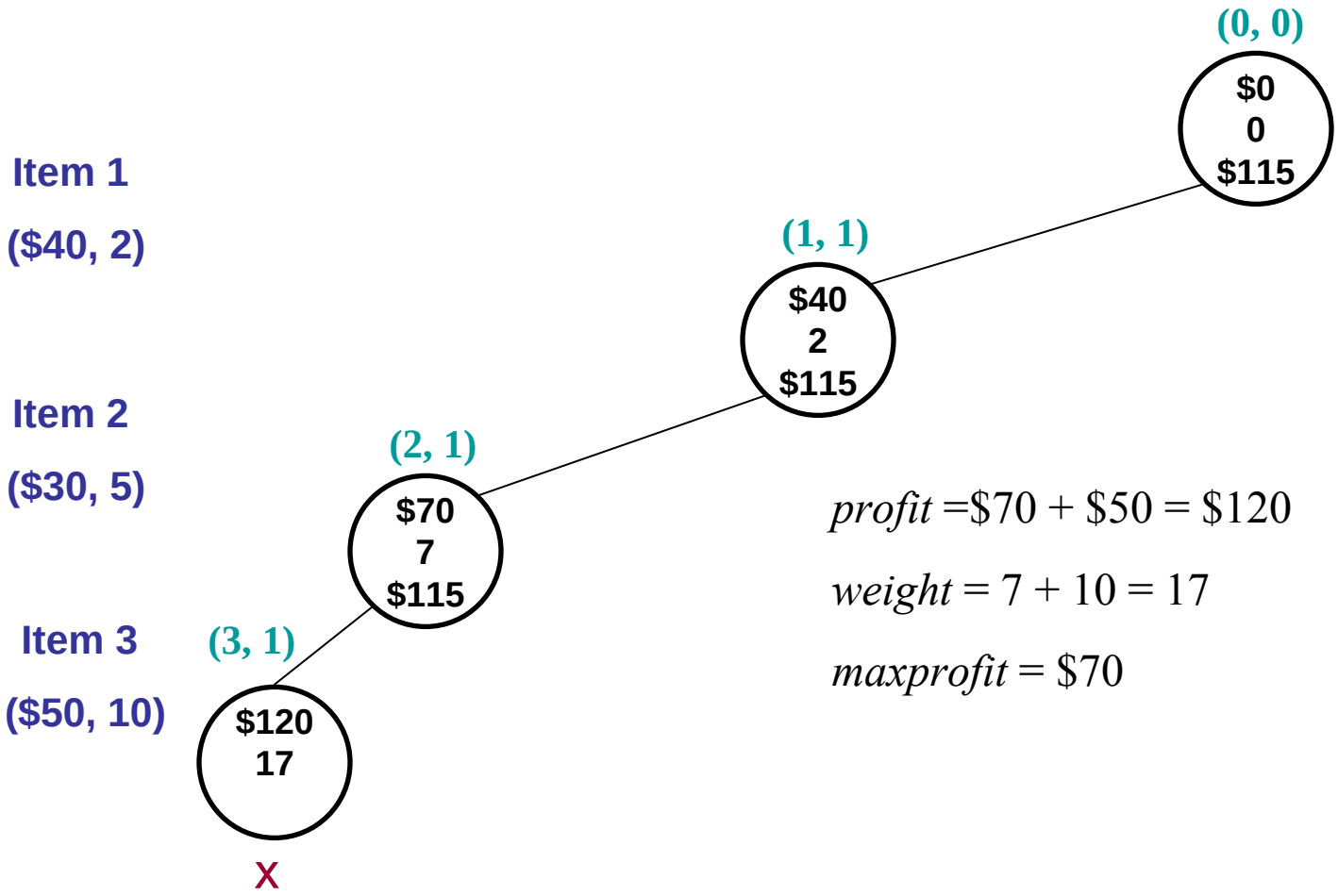
$$maxprofit = \$70$$

$$totoweight = weight + \sum_{j=2+1}^{3-1} w_j = 7$$

$$bound = \$70 + (16 - 7) * \frac{\$50}{10} = \$115$$

این گره امید بخش می باشد زیرا وزنش کمتر از ۱۶ (W) و حدش بزرگتر از $maxprofit = 70$ می باشد.

مثال



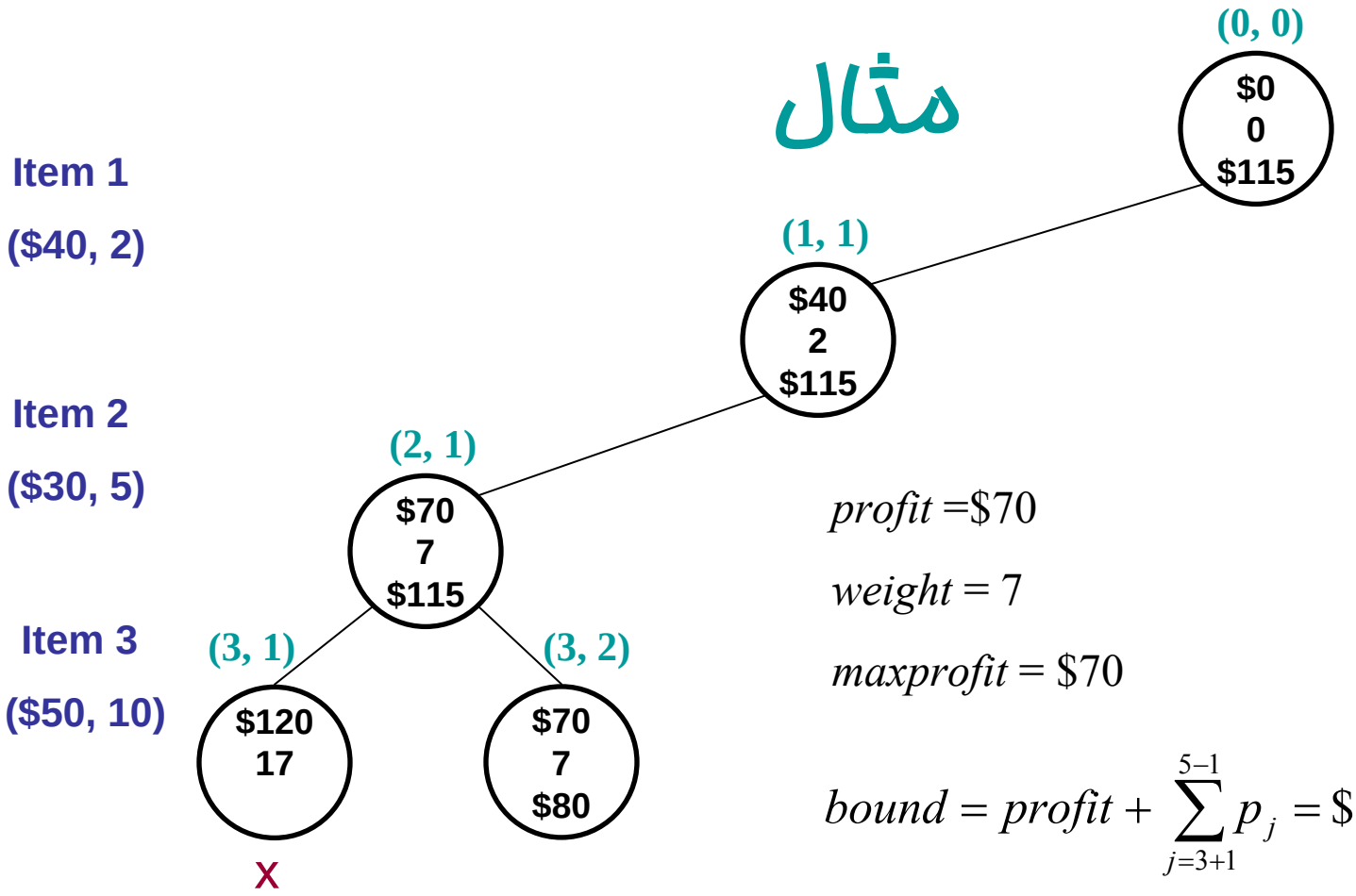
$profit$ =\$70 + \$50 = \$120

$weight$ = 7 + 10 = 17

$maxprofit$ = \$70

این گره امید بخش نمی باشد زیرا وزنش بیشتر از ۱۶ (W) است و بنابراین حدش را محاسبه نمی کنیم

مثال



$$profit = \$70$$

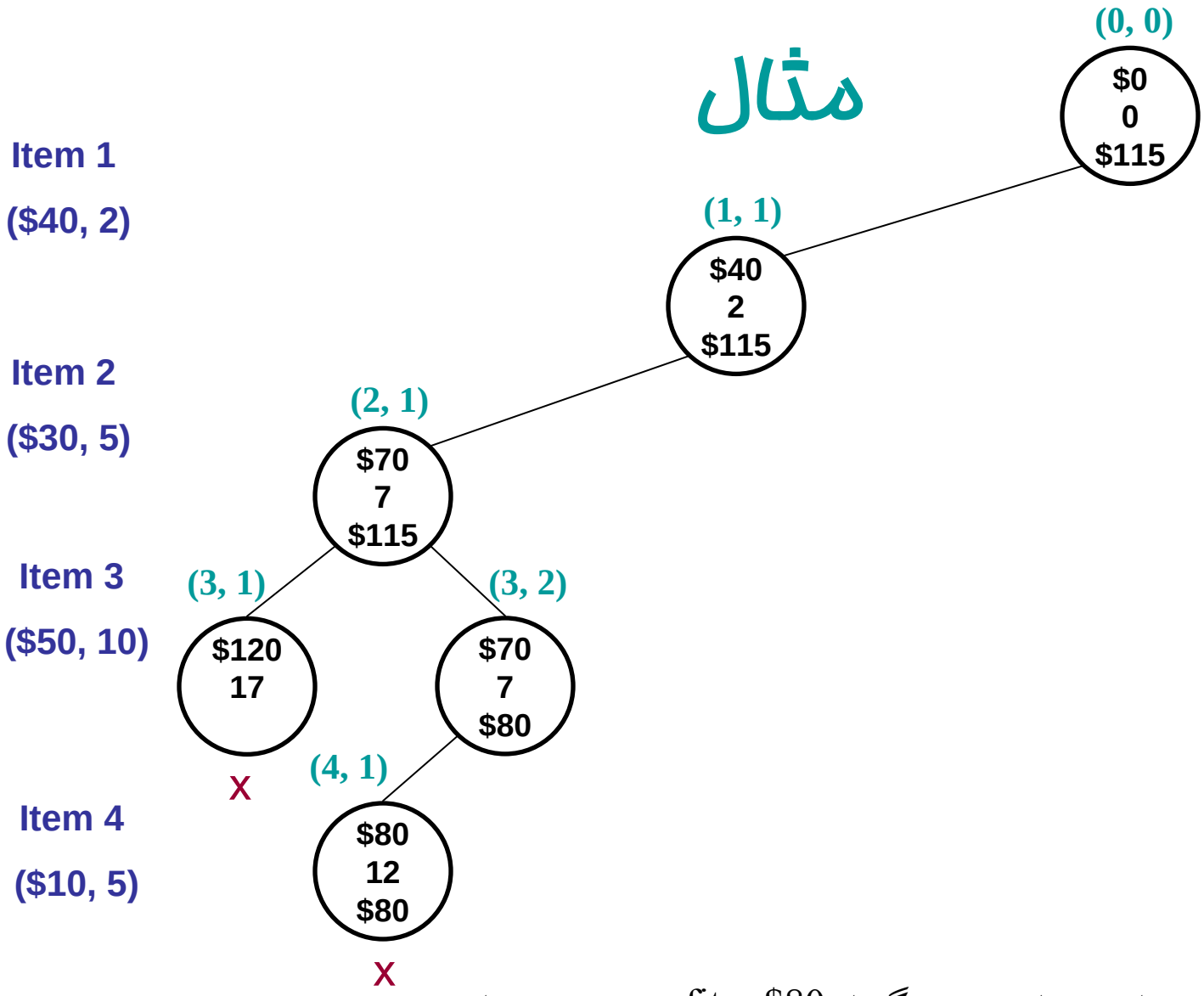
$$weight = 7$$

$$maxprofit = \$70$$

$$bound = profit + \sum_{j=3+1}^{5-1} p_j = \$70 + \$10 = \$80$$

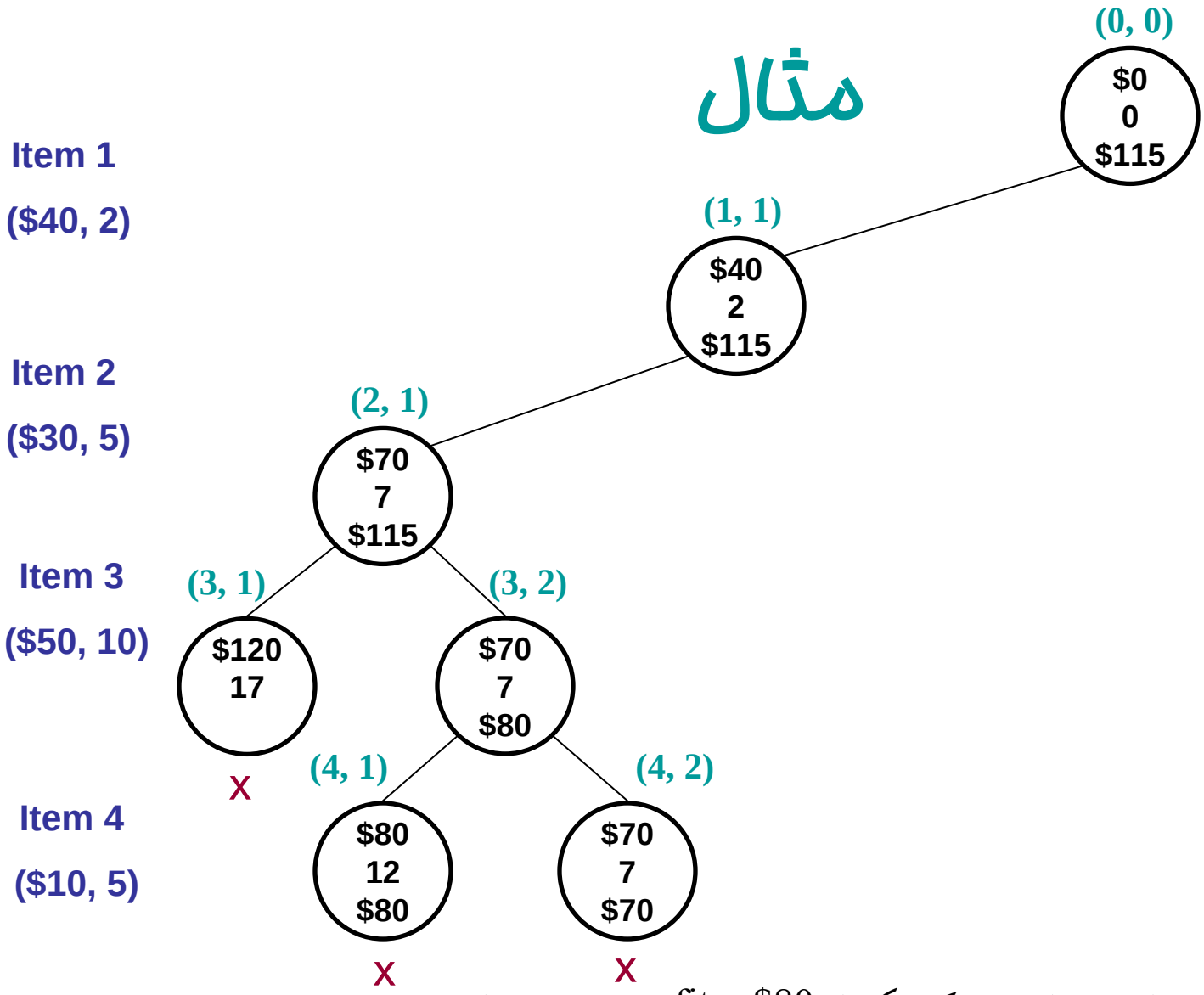
این گره امید بخش می باشد زیرا وزنش کمتر از ۱۶ (W) است و حدش بزرگتر از maxprofit = $70 می باشد

مثال



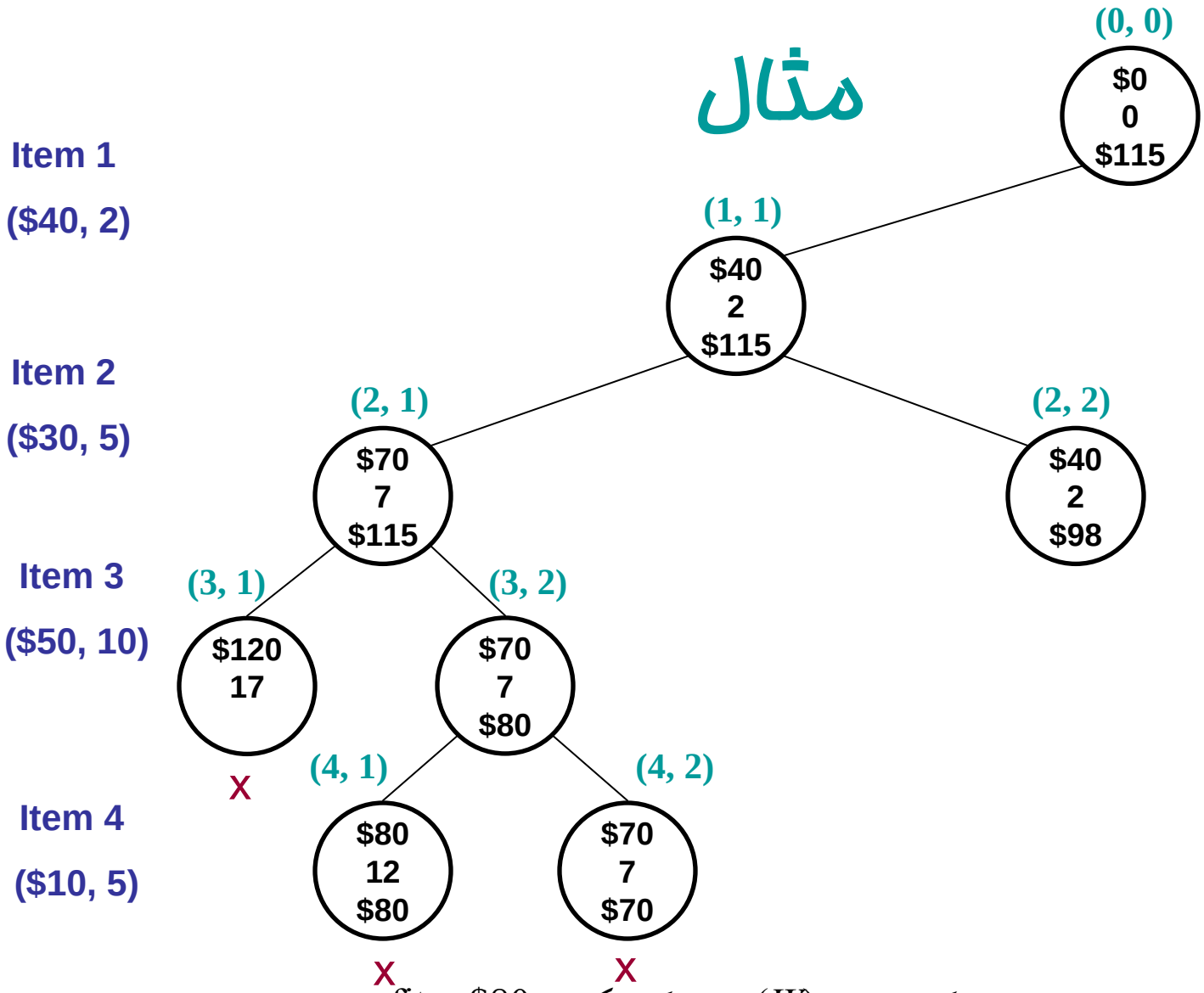
این گره امید بخش نمی باشد زیرا حدش بزرگتر از maxprofit = $80 نمی باشد

مثال



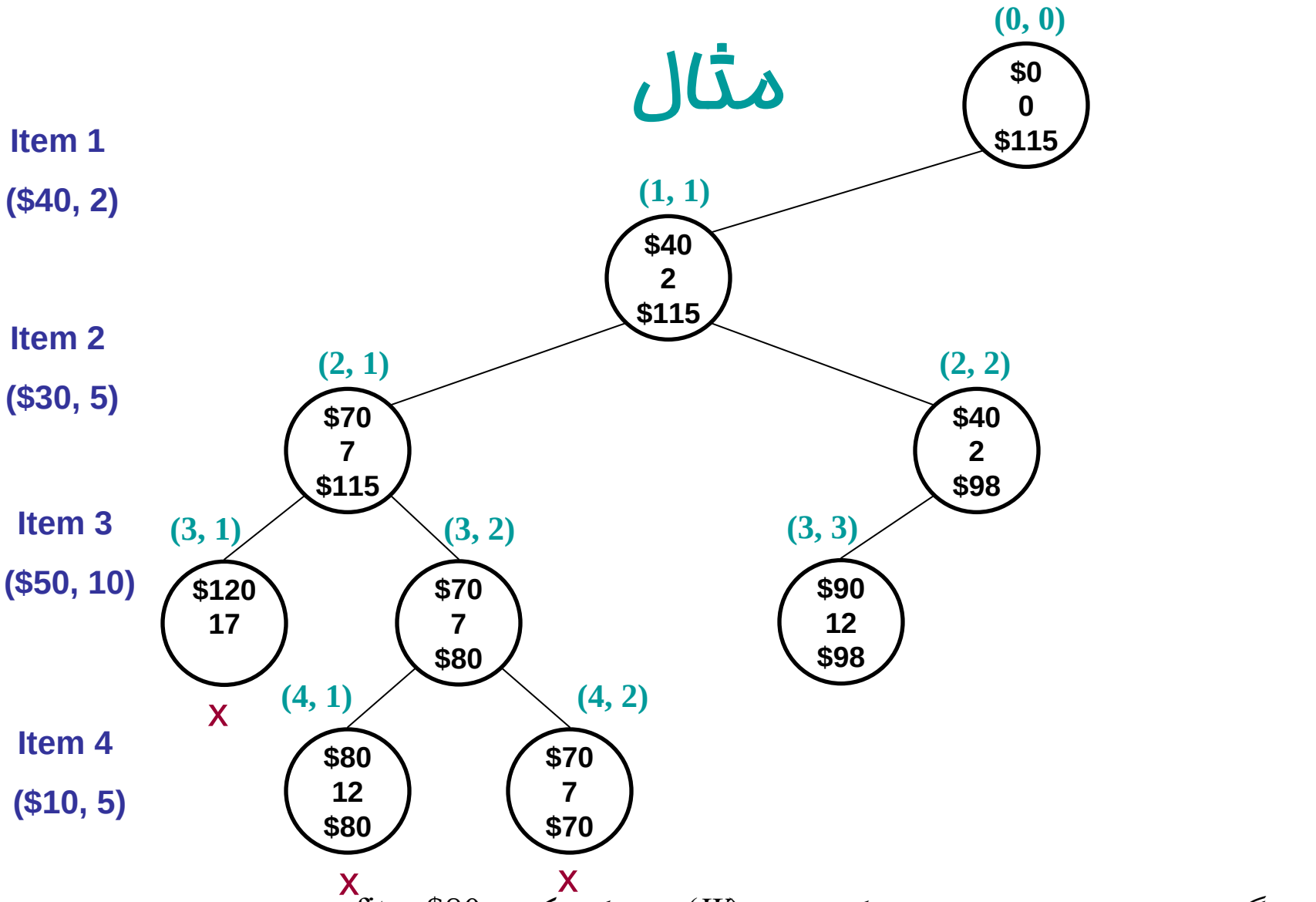
این گره امید بخش می باشد زیرا حدش کوچکتر از maxprofit = $80 می باشد

مثال



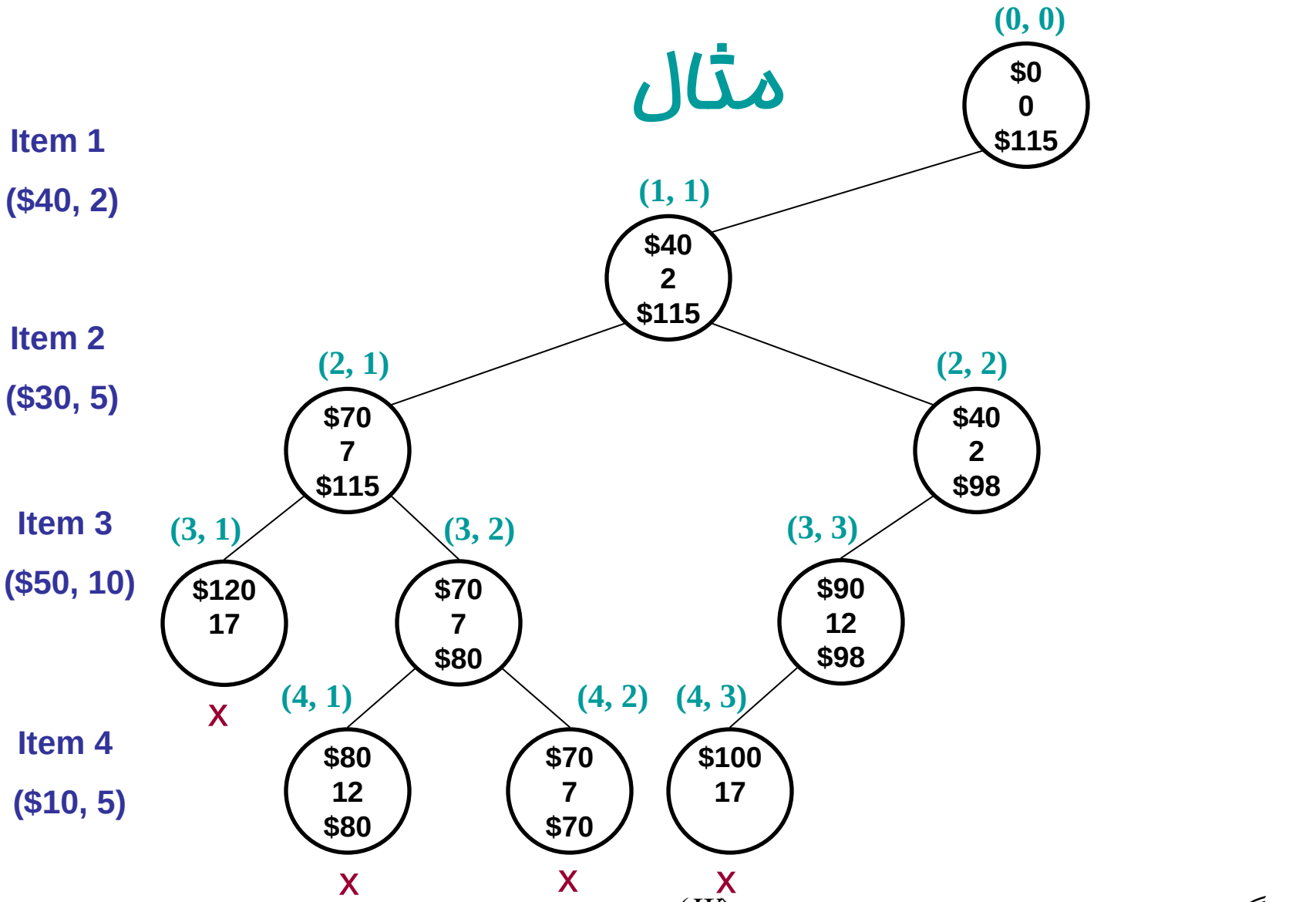
این گره امید بخش می باشد زیرا وزنش کمتر از ۱۶ (W)حدش کوچکتر از $80 = maxprofit می باشد

مثال



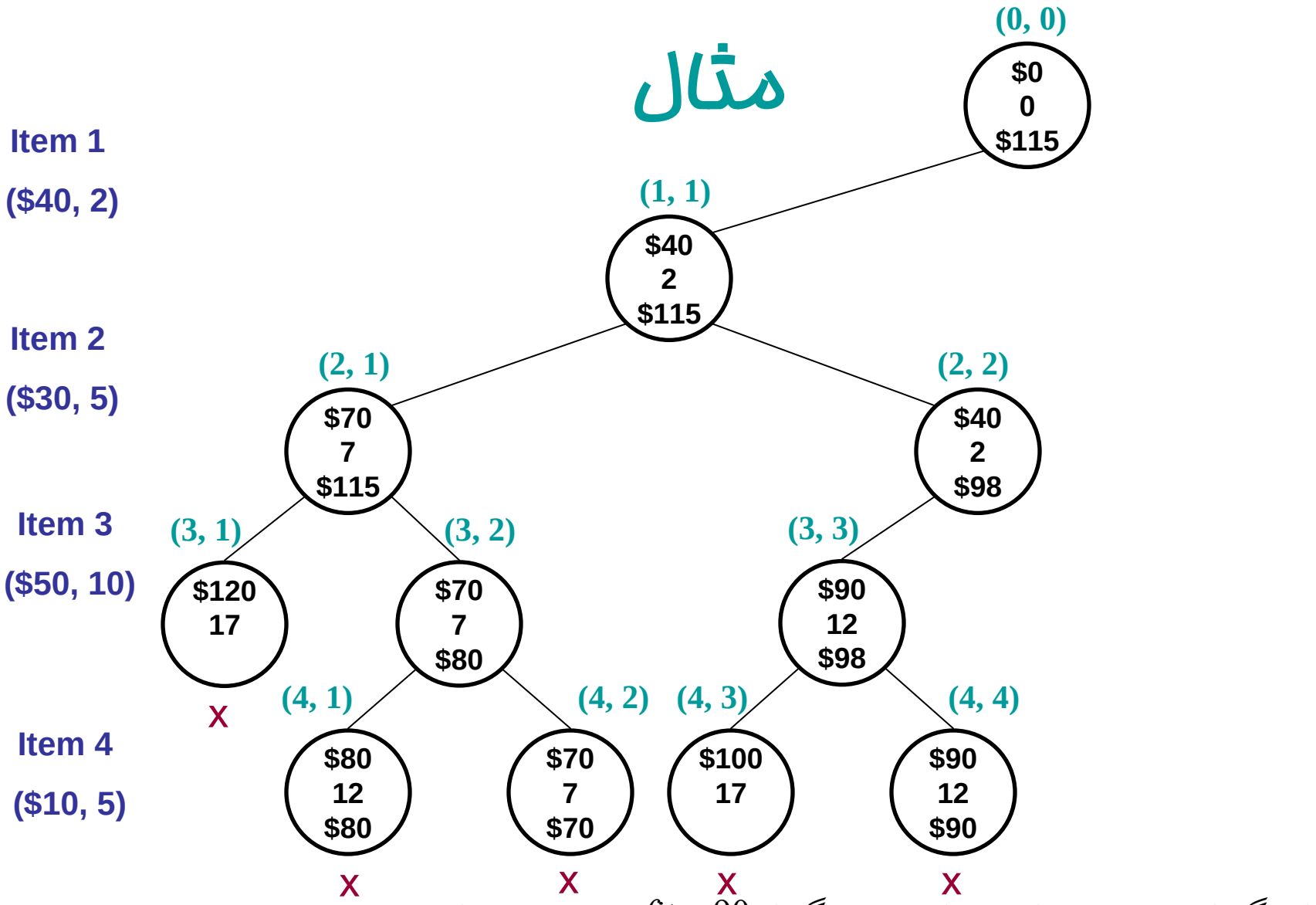
این گره امید بخش می باشد زیرا وزنش کمتر از ۱۶ (W)حدش کوچکتر از maxprofit = $80 می باشد

مثال



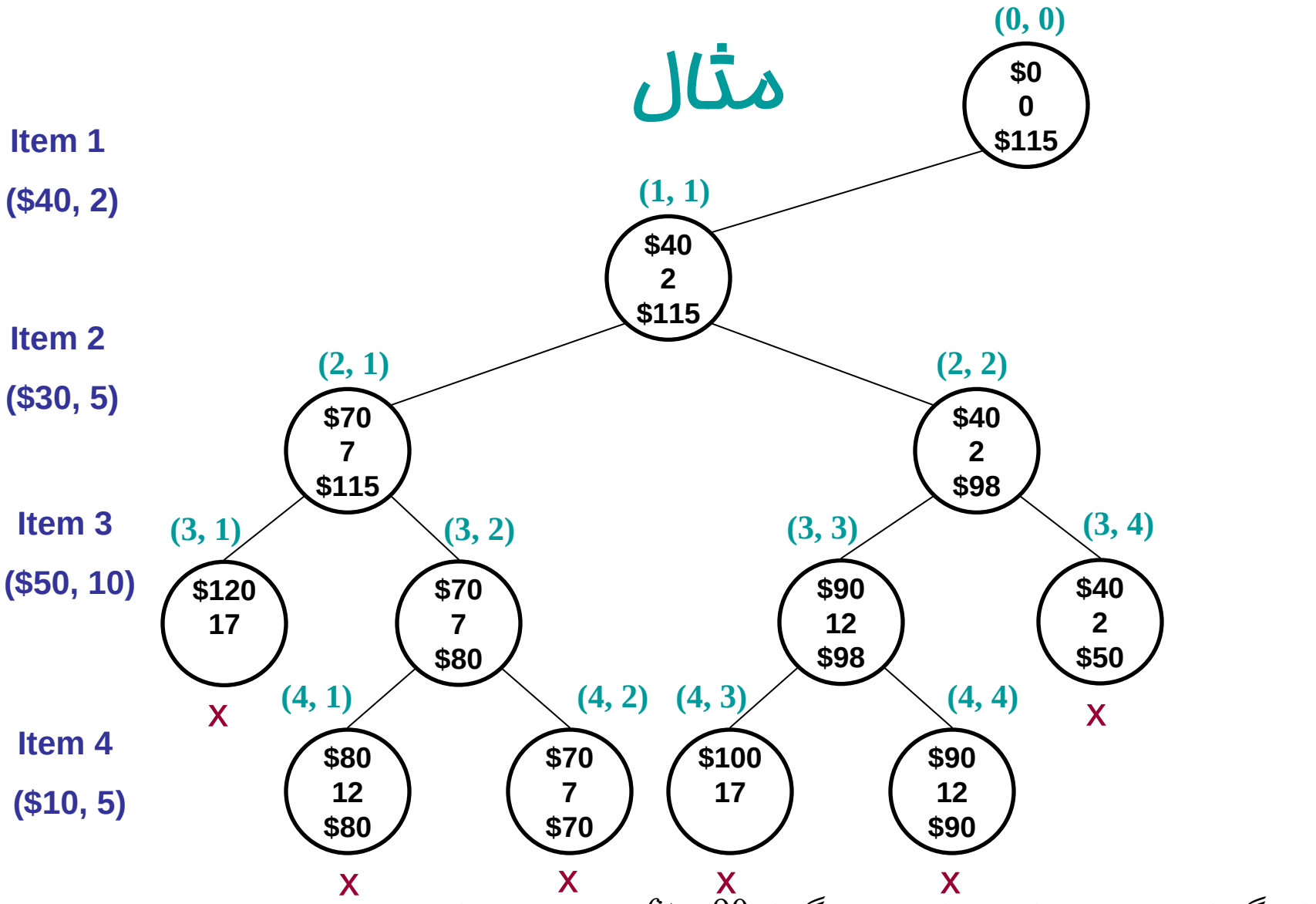
این گره امید بخش نمی باشد زیرا وزنش بیشتر از ۱۶ (W)و بنابراین حدش محاسبه نمی شود

مثال



این گره امید بخش نمی باشد زیرا حدش بزرگتر از $maxprofit = 90$ نمی باشد

مثال



این گره امید بخش نمی باشد زیرا حدش بزرگتر از $maxprofit = 90$ نمی باشد

الگوریتم

- مساله: n قطعه که هر یک دارای وزن و ارزش مشخصی می باشد، داده شده است. وزن و ارزش هر قطعه یک عدد صحیح و مثبت می باشد. علاوه براین، عدد صحیح و مثبت W داده شده است. مطلوب است تعیین مجموعه ای از قطعات با حداکثر ارزش به شرط آن که حاصل جمع اوزان آنها از W بیشتر نباشد.

- ورودی ها: اعداد صحیح و مثبت n و W. آرایه های w و p که هر کدام از ۱ تا n اندیس گذاری شده اند و حاوی اعداد صحیح و مثبتی می باشند که که بر اساس مقادیر p_i / w_i به صورت غیر نزولی مرتب شده اند.

- خروجی ها: آرایه $bestset$ که از ۱ تا n اندیس گذاری شده است و در آن مقدار $bestset[i]$ در صورتی "yes" می باشد که قطعه i اُم در مجموعه بهینه گنجانده شود. عدد صحیح $maxprofit$ که ارزش بیشینه را نشان می دهد.

الگوریتم

```
void knapsack ( index i, int profit, int weight )
{
    if ( weight <= W && profit > maxprofit ) {
            maxprofit = profit ;
            numbest = i ;
            bestset = include ;
    }
    if ( promising (i)) {
            include [i + 1] = "yes" ;
            knapsack (i + 1, profit + p [i + 1], weight + w [i + 1] ) ;
            include [i + 1] = "no" ;
            knapsack ( i + 1, profit, weight) ;
    }
}
```

الگوریتم

```
bool promising ( index i)
{
    index j, k ;
    int totweight ;
    float bound ;

    if ( weight >= W)
        return false ;
    else {
        j = i + 1 ;
        bound = profit ;
        totweight = weight ;
        while ( j <= n && totweight + w [j] <= W) {
            totweight = totweight + w [j] ;
            bound = bound + p [j];
            j++;
        }
        k = j;
        if ( k <= n)
            bound = bound + ( W – totweight * p [k] / w [k] ;
        return bound > maxprofit ;
    }
}
```

پیچیدگی زمانی

- تعداد گره های درخت فضای حالت 2^{n+1}-1
- مثال از بدترین حالت

$$p_i = 1 \qquad w_i = 1 \qquad 1 \le i \le n-1$$

$$p_n = n \qquad w_n = n$$

مقایسه الگوریتم برنامه ریزی پویا و عقبگرد برای مساله کوله پشتی ۰-۱

- الگوریتم برنامه نویسی پویا در بدترین حالت

$$O(minimum(2^n, nW))$$

- الگوریتم عقبگرد

$$\Theta(2^n)$$

- هورویتز و ساهنی نشان داده اند که الگوریتم عقبگرد معمولا نسبت به الگوریتم برنامه ریزی پویا کارآیی بیشتری دارد.
- ترکیب روش تقسیم و حل و برنامه ریزی پویا توسط هورویتز و ساهنی برای حل مساله کوله پشتی ۰-۱
  - در بدترین حالت $O(2^{n/2})$
  - این الگوریتم معمولا کارآیی بیشتری نسبت به عقب گرد دارد.