به نام خدا

فصل ششم

راهبرد شاخه و حد

Branch-and-Bound

دانشکده مهندسی کامپیوتر – دانشگاه اصفهان
دکتر مرجان کائدی

الگوریتم شاخه و حد

- الگوریتم شاخه و حد شکل بهبود یافته ای از الگوریتم عقبگرد است.
- در هر دو برای حل مساله از درخت جستجو استفاده می شود.
- ولی روش شاخه و حد فقط برای مسائل بهینه سازی به کار می رود و در آن، به پیمایش خاصی از درخت محدود نیستیم.

- در الگوریتم شاخه و حد، برای هر گره درخت، حدی را محاسبه می کنیم تا تعیین شود که آن گره امیدبخش است یا خیر.
- این عدد، حدی برای مقدار حل است که با گسترش یافتن آن گره قابل دستیابی است.
- اگر حد یک گره بهتر از مقدار بهترین حلی که تا کنون یافته شده است نباشد، گره غیرامیدبخش است.

الگوریتم شاخه و حد

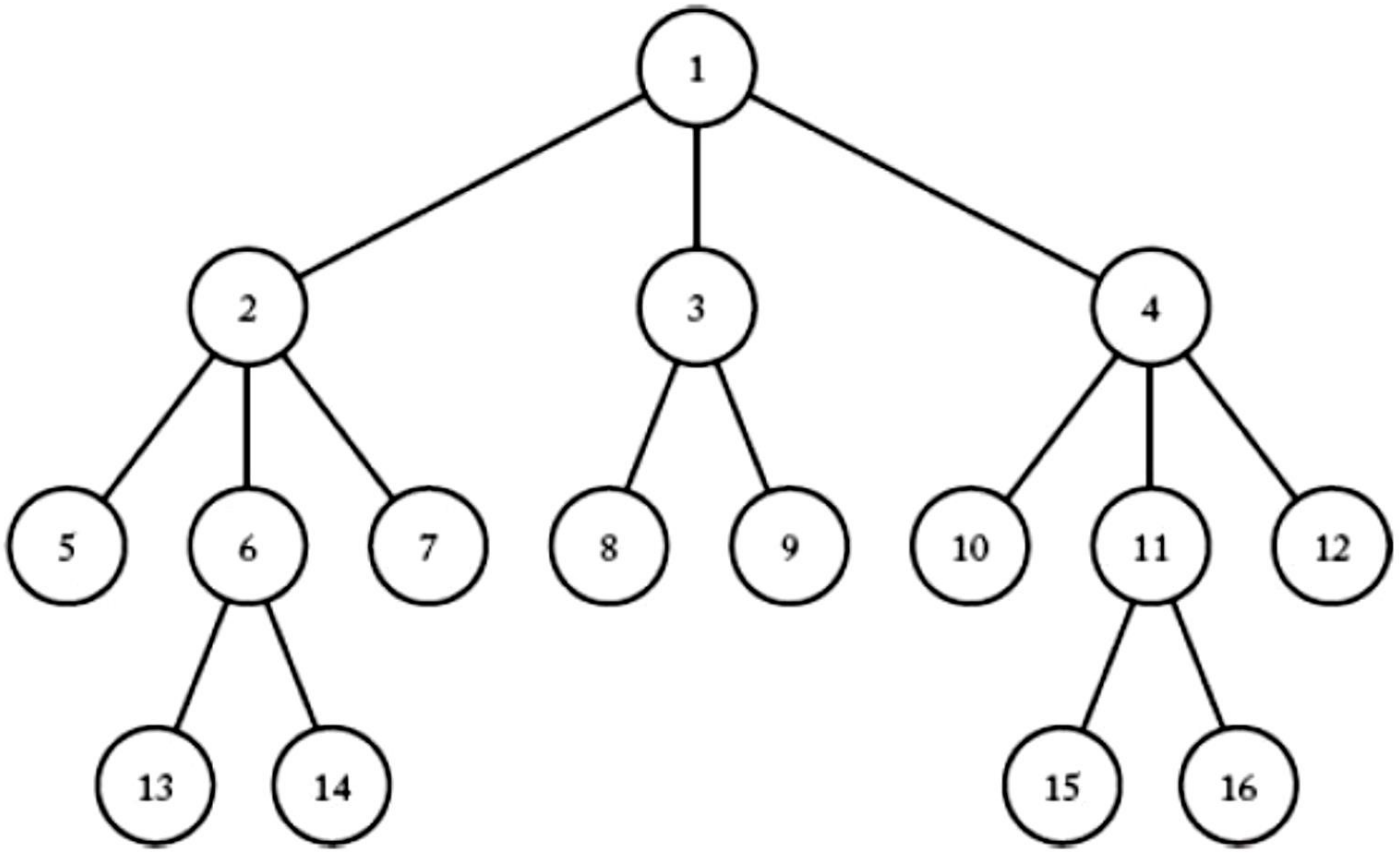
- زمان الگوریتم شاخه و حد نیز مانند الگوریتم عقبگرد، در بدترین حالت زمانی نمایی (یا بدتر از نمایی) است.
- ولی به هر حال می تواند برای بسیاری از نمونه های بزرگ مساله موثر باشد.

در ادامه دو نوع الگوریتم معرفی می شوند:

1. جستجوی عرضی هرس کردن شاخه و حد
2. بهترین–جستجو با هرس کردن شاخه و حد (نسخه اصلاح شده ی روش ۱)

- ابتدا مفهوم جستجوی عرضی را مرور خواهیم کرد.
- سپس به الگوریتم جستجوی عرضی، هرس کردن شاخه های غیرامیدبخش را هم اضافه می کنیم.
- بعد خواهیم دید که می توان به جای انجام جستجوی عرضی، حدهای گره ها را با هم مقایسه کرد و فرزندان گره هایی با بهترین حد را ملاقات کرد (به جای ملاقات گره ها بر طبق یک ترتیب از پیش تعیین شده در جستجوی عرضی)

جستجوی عرضی (breadth first) در یک درخت



جستجوی عرضی در یک درخت

```
void breadth_first_tree_search (tree T);
{
 queue_of_node Q;
 node u, v,
 initialize (Q);                    // Intialize Q to be empty.
 v = root of T;
 visit v;

 enqueue (Q, v);
 while (! empty (Q)) {
   dequeue (Q, v);
   for (each child u of v){
      visit u;
      enqueue (Q, u);
   }
 }
}
```

الگوریتم جستجوی عرضی با هرس کردن شاخه و حد در حالت کلی

```
void breadth_first_branch_and_bound(state_space_tree T, number& best)
{
 queue_of_node Q;
 node u, v,

 initialize (Q);          // Initialize Q to be empty.
 v = root of T;           // Visit root.
 enqueue (Q, v);
 best = value (v);

 while (! empty (Q)){
   dequeue (Q, v);
   for (each child u of v){   // Visit each child.
       if (value (u) is better than best)
           best = value (u);
       if (bound (u) is better than best)
           enqueue (Q, u);
   }
 }
}
```

الگوریتم شاخه و حد برای مساله کوله پشتی صفر و یک

مساله کوله پشتی صفر و یک

- مساله کوله پشتی صفر و یک در فصل چهارم معرفی شد.
- دیدیم که به عنوان یک رهیافت حریصانه می توان اشیاء را بر اساس نسبت ارزش به وزنشان مرتب کرد و یکی یکی انتخاب کرد و در کوله پشتی قرار داد.

- ولی آن رهیافت لزوما بهترین جواب را به دست نمی آورد.

جستجوی عرضی با هرس کردن شاخه و حد برای کوله پشتی صفر و یک

- **مثال:**

$n = 4,\ W = 16$

- حد گره (bound): مرز بالای مقدار سودی که می توان از گسترش دادن گره به دست آورد (این حد را با حل مساله کوله پشتی پیوسته تخمین می زنیم.)
- weight و profit : جمع وزن ها و جمع منفعت های به دست آمده تا این گره از درخت

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

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

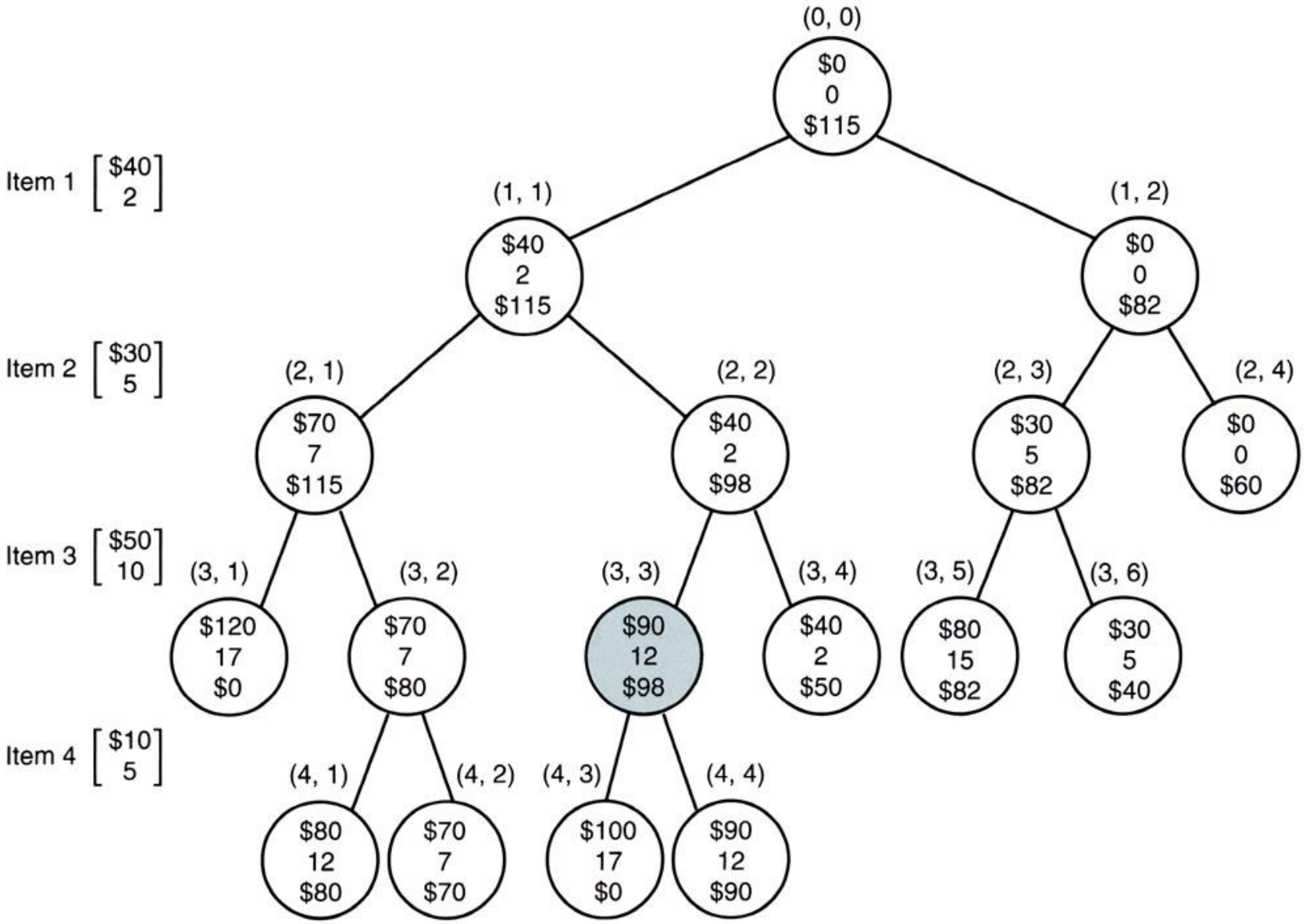
$$bound = \left(profit + \sum_{j=i+1}^{k-1} p_j \right) + (W - totweight) \times \frac{p_k}{w_k}.$$

- گره های درخت را ارزیابی می کنیم.
- در اینجا یک گره در صورتی امیدبخش است که این حد کوچکتر یا مساوی maxprofit یعنی مقدار بهترین حلی باشد که تا آن هنگام پیدا شده است.
- علاوه بر آن اگر شرط زیر برای یک گره برقرار باشد، ادامه دادن آن گره امیدبخش نیست:

$$weight \geq W.$$

- **الف) الگوریتم جستجوی عرضی با هرس کردن شاخه و حد برای مساله کوله پشتی صفر و یک**

- اطلاعات نوشته شده در هر گره به ترتیب:
  - Profit
  - weight
  - Bound



$n = 4,\ W = 16$

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

محاسبه حد (bound) برای هر گره

- حد هر گره برابر است با ارزشی که تا کنون در آن گره به صورت واقعی به دست آمده است به اضافه حداکثر ارزش تخمینی که از ادامه دادن آن گره به دست خواهد آمد.

الگوریتم جستجوی عرضی با هرس کردن شاخه و حد

- این الگوریتم نوعی پیمایش عرضی درخت است.
- ولی در این پیمایش، یک گره درخت تنها در صورتی گسترش می یابد که حد آن بهتر از مقدار بهترین حل فعلی باشد.

- در شروع کار، مقدار اولیه بهترین حل فعلی، برابر است با مقدار حل در ریشه و یا یک مقدار اولیه که بدتر از هر حلی باشد (برای مسائلی که در ریشه جوابی ندارند)

الگوریتم جستجوی عرضی با هرس کردن شاخه و حد

```
void breadth_first_branch_and_bound
(state_space_tree T,
number& best)
{queue_of_node Q;
 node u, v,

 initialize (Q);          // Initialize Q to be empty.
 v = root of T;           // Visit root.
 enqueue (Q, v);
 best = value (v);

 while (! empty (Q)){
   dequeue (Q, v);
   for (each child u of v){   // Visit each child.
       if (value (u) is better than best)
           best = value (u);
       if (bound (u) is better than best)
           enqueue (Q, u);
   }
 }
}
```

تابع محاسبه حد یک گره برای مساله کوله پشتی

```
float bound (node u)
{
  index j, k;
  int totweight;
  float result;

  if (u. weight >= W)
      return o;
  else{
      result = u. profit;
      totweight = u. weight;
      j = u. level + 1;
     while (j <= n && totweight + w[j] <= W){
          totweight = totweight + w[j];        // Grab as many items as possible.
          result = result + p[j];
          j++;
      }
      k = j;                          // Use with formula in text.

     if (k <=n)
        result = result + (W - totweight) * p[k] /w[k];      // Grab fraction of kth item.

     return result; }
```

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

$$bound = \left(profit + \sum_{j=i+1}^{k-1} p_j \right) + (W - totweight) \times \frac{p_k}{w_k}.$$

- می توان الگوریتم را به نحوی اصلاح کرد که در هر گره متغیری به نام items هم داشته باشیم که حاوی مجموعه اشیائی است که تا آن گره لحاظ شده اند.

- و متغیر bestitems را هم تعریف می کنیم که با بهترین مجموعه اشیاء فعلی مقداردهی می شود (در همان زمانی که مقدار best به روز می شود).

- **ب) الگوریتم بهترین جستجو با هرس کردن شاخه و حد برای مساله کوله پشتی صفر و یک**

- می توانیم جستجو را بهبود دهیم تا علاوه بر تعیین امیدبخش بودن یک گره، کار دیگری را هم انجام دهد.
- به این صورت که می توانیم پس از ملاقات همه فرزندان یک گره مفروض، همه گره های امیدبخش و گسترش نیافته را بررسی کنیم و آن را که بهترین حد را دارد را گسترش دهیم.

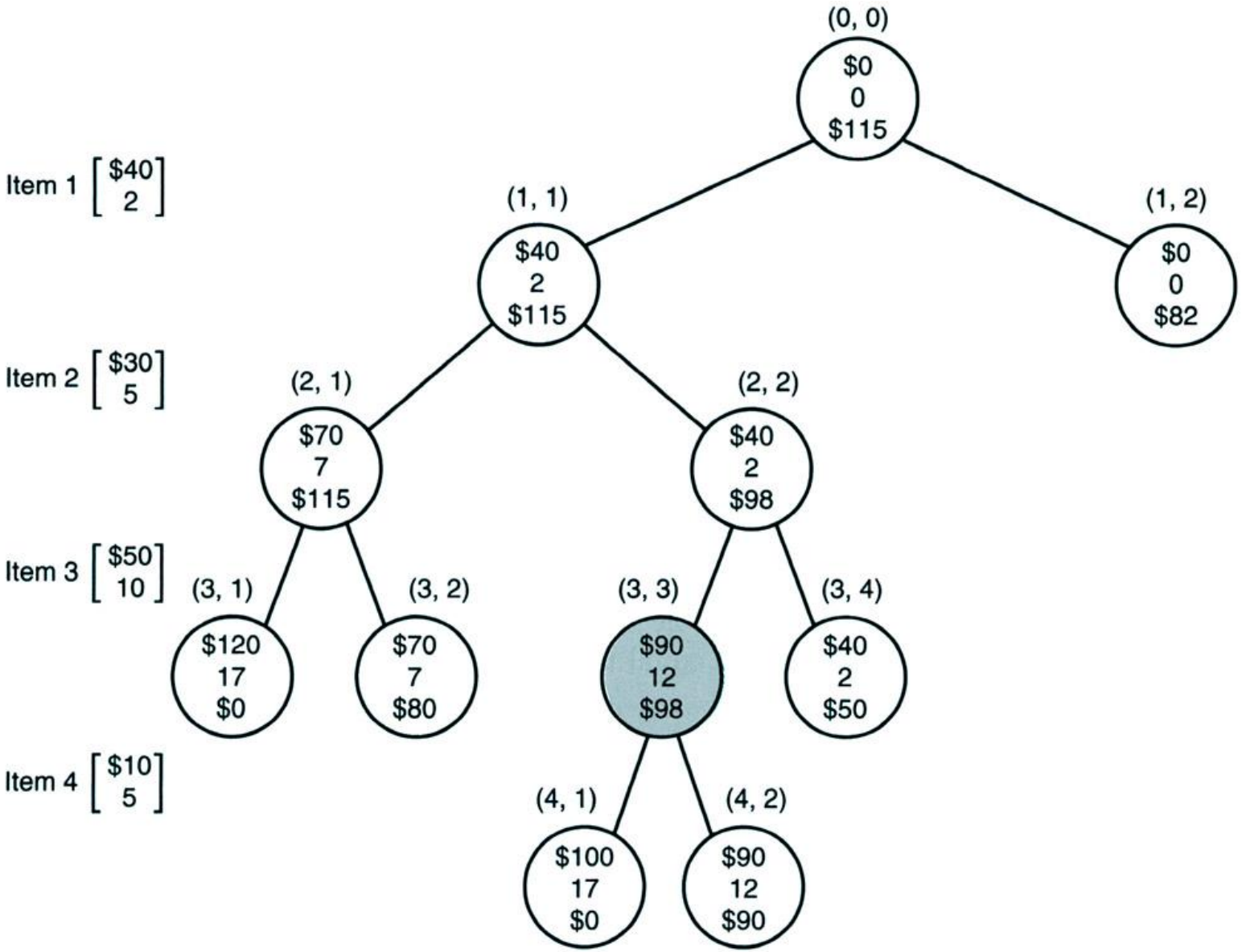
- به این ترتیب، نسبت به حالتی که صرفا کورکورانه بر طبق یک ترتیب از پیش تعیین شده جلو می رفتیم، سریع تر به حل بهینه می رسیم.

الگوریتم بهترین جستجو با هرس کردن شاخه و حد در حالت کلی

```
void best_first_branch_and_bound ( state_space_tree T,
                    number& best
{
 priority_queue_of_node PQ;
 node u, v;

 initialize (PQ);              // Initialize PQ to be empty.
 v = root of T;
 best = value(v);
 insert (PQ, v);

 while (! empty (PQ)) {          // Remove node with best bound.
    remove (PQ, v);
    if (bound (v) is better than best)      // Check if node is still promising.
        for (each child u of v) {
            if (value(u) is better than best)
                best = value (u);
            if (bound (u) is better than best)
                insert (PQ, u);
        }
 }
}
```



$n = 4, W = 16$

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

- با استفاده از بهترین جستجو، فقط ۱۱ گره را چک کردیم که ۶ گره کمتر از تعداد گره های چک شده در جستجوی عرضی است.

- در درخت های فضای حالت بزرگ تر، تعداد گره های کمتر در اثر استفاده از بهترین جستجو برای یافتن حل بهینه چشمگیر می شود.

- تضمینی وجود ندارد که گره ای که بهتر به نظر می رسد، منجر به حل بهینه شود.
- در پیاده سازی الگوریتم بهترین جستجو، به جای استفاده از صف معمولی، از صف اولویت استفاده می کنیم.

- علاوه بر استفاده از صف اولویت، پس از حذف یک گره از صف اولویت، شرطی اضافه شده است که تعیین می کند آیا حد مربوط به گره هنوز بهتر از best است یا خیر.
- به این ترتیب، تعیین می شود که آیا گره ای بعد از ملاقات شدن، همچنان امیدبخش است یا خیر.
- این موضوع را از طریق مقایسه حد آن گره با maxprofit پس از حذف آن از صف اولویت می توان دریافت.

- به این ترتیب، از ملاقات فرزندان گره ای که پس از ملاقات شدن، غیرامیدبخش می شود، پرهیز می کنیم.

مساله فروشنده دوره گرد

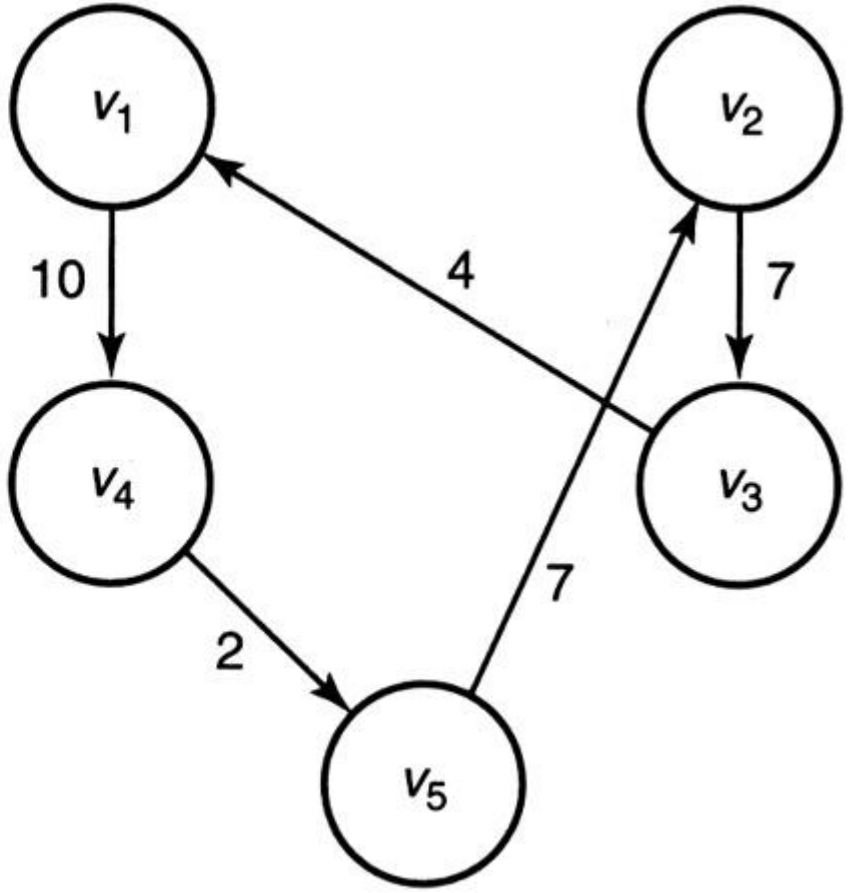
Travelling Salesman Problem (TSP)

- در فصل قبل از الگوریتم عقبگرد برای یافتن دور هامیلتونی در گراف استفاده شد که البته ممکن بود دور یافته شده، بهینه نباشد و با دور بهینه خیلی فاصله داشته باشد.

- هدف مساله فروشنده دوره گرد: یافتن کوتاه ترین مسیر در یک گراف جهت دار با شروع از یک راس مفروض است، مشروط بر آن که هر راس فقط یکبار ملاقات شود.

ماتریس مجاورت یک گراف و تور بهینه مربوط به آن

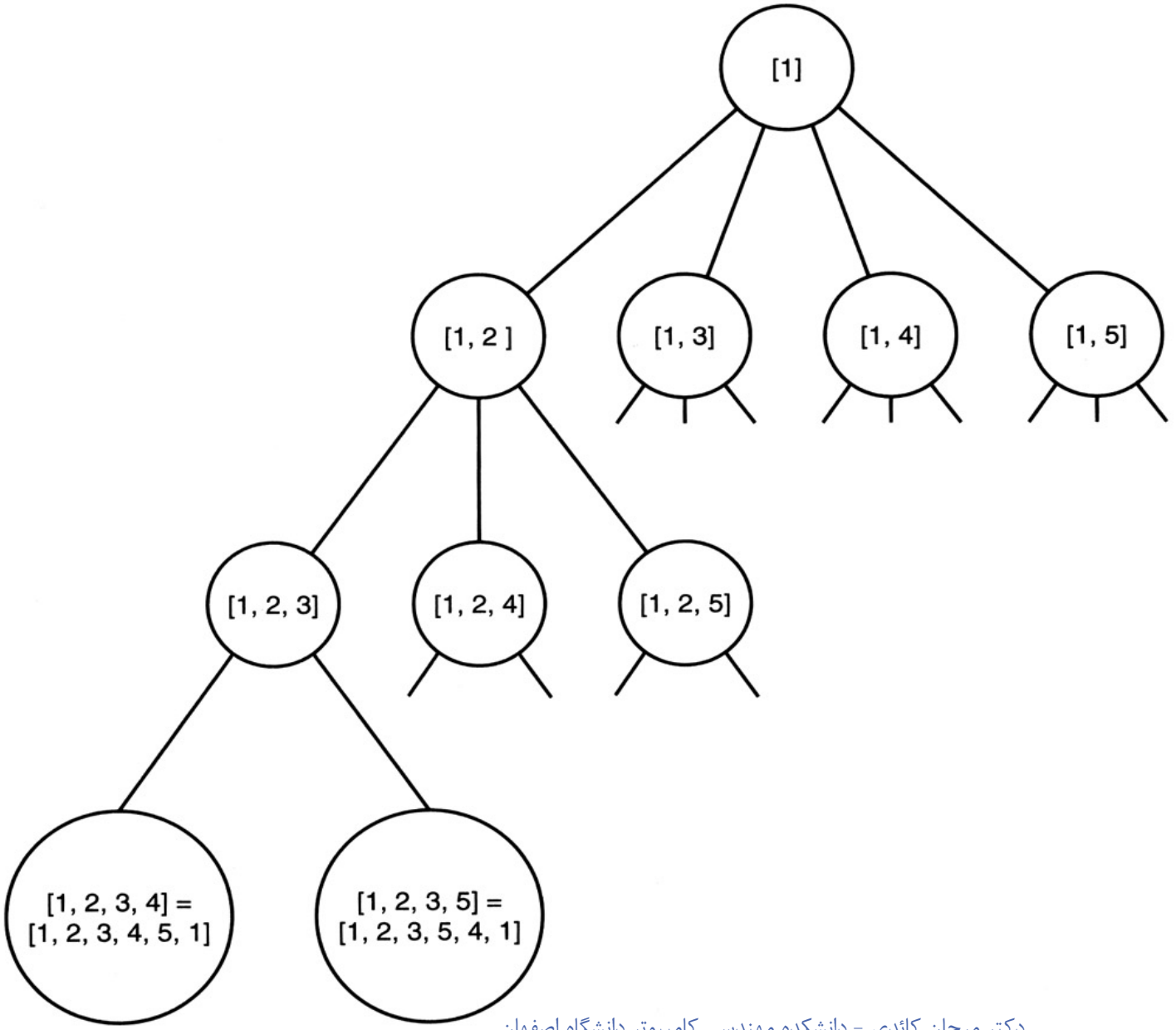
$$\begin{bmatrix} 0 & 14 & 4 & 10 & 20 \\ 14 & 0 & 7 & 8 & 7 \\ 4 & 5 & 0 & 7 & 16 \\ 11 & 7 & 9 & 0 & 2 \\ 18 & 7 & 17 & 4 & 0 \end{bmatrix}$$



- در درخت، گره ای که برگ نباشد نشانگر همه تورهایی است که با مسیر نگهداری شده در آن گره آغاز می شود.

- مثلا گره حاوی [1,2,3] نشانگر همه تورهایی است که با مسیر [1,2,3] آغاز می شوند.

- برای مثال مذکور، هنگامیکه چهار راس از گراف در یک گره نگهداری می شوند، از توسعه درخت باز می ایستیم زیرا راس پنجم مشخص است.
- در این مساله باید یک حد پایینی برای طول هر تور بیابیم که از گسترش دادن یک گره مفروض قابل حصول باشد و گره را فقط در صورتی امیدبخش بخوانیم که حد آن کوچکتر از طول کمینه فعلی باشد.


[1]
[1, 2]
[1, 3]
[1, 4]
[1, 5]
[1, 2, 3]
[1, 2, 4]
[1, 2, 5]
[1, 2, 3, 4] =
[1, 2, 3, 4, 5, 1]
[1, 2, 3, 5] =
[1, 2, 3, 5, 4, 1]

محاسبه حد پایین برای هر گره

- مساله فروشنده دوره گرد یک مساله کمینه سازی است.
- برای به دست آوردن حد پایین برای هر گره، به این صورت عمل می کنیم:
- در هر تور، طول یال انتخاب شده برای خروج از هر راس گراف، باید حداقل به بزرگی طول کوتاه ترین یال خارج شده از آن راس باشد.
- بنابراین:
- حد پایین هزینه برای خروج از راس V1 عبارت است از مقدار مینیمم همه عناصر غیرصفر در سطر ۱ از ماتریس مجاورت گراف
- حد پایین هزینه برای خروج از راس V2 عبارت است از مقدار مینیمم همه عناصر غیرصفر در سطر ۲ از ماتریس مجاورت گراف
- و الی آخر....

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

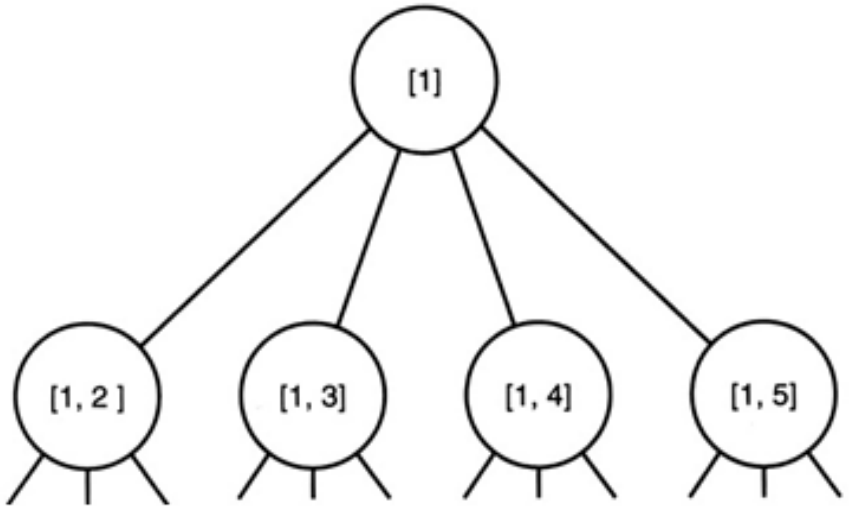
- بنابراین داریم:

$$\begin{aligned} v_1 \quad & minimum\,(14, 4, 10, 20) = 4 \\ v_2 \quad & minimum\,(14, 7, 8, 7) = 7 \\ v_3 \quad & minimum\,(4, 5, 7, 16) = 4 \\ v_4 \quad & minimum\,(11, 7, 9, 2) = 2 \\ v_5 \quad & minimum\,(18, 7, 17, 4) = 4 \end{aligned}$$

- حد پایین طول تور، برابر حاصل جمع این مقادیر مینیمم است.

$$4 + 7 + 4 + 2 + 4 = 21.$$

- این بدین معنا نیست که توری با این طول در واقع وجود دارد، بلکه به این معنا است که توری با طول کوتاه تر از این وجود ندارد.



- به دست آوردن حد پایین برای طول دور، پس از انتخاب گام دوم دور (نودی که در مرحله قبل انتخاب شده است را این بار در مینیمم گیری ها به حساب نمی آوریم):

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

$$\begin{aligned} v_1 & & 14 \\ v_2 & \ minimum\ (7,8,7) & = 7 \\ v_3 & \ minimum\ (4,7,16) & = 4 \\ v_4 & \ minimum\ (11,9,2) & = 2 \\ v_5 & \ minimum\ (18,17,4) & = 4 \end{aligned}$$

- حد پایین:

$$14 + 7 + 4 + 2 + 4 = 31.$$

- به دست آوردن حد پایین برای طول دور، پس از انتخاب گام سوم دور (نودهایی که در مرحله قبل انتخاب شده اند را این بار در مینیمم گیری ها به حساب نمی آوریم):

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

$$\begin{aligned} v_1 & & 14 \\ v_2 & & 7 \\ v_3 & minimum\,(7,16) = & 7 \\ v_4 & minimum\,(11,2) = & 2 \\ v_5 & minimum\,(18,4) = & 4 \end{aligned}$$

- حد پایین:

$$14 + 7 + 7 + 2 + 4 = 34.$$

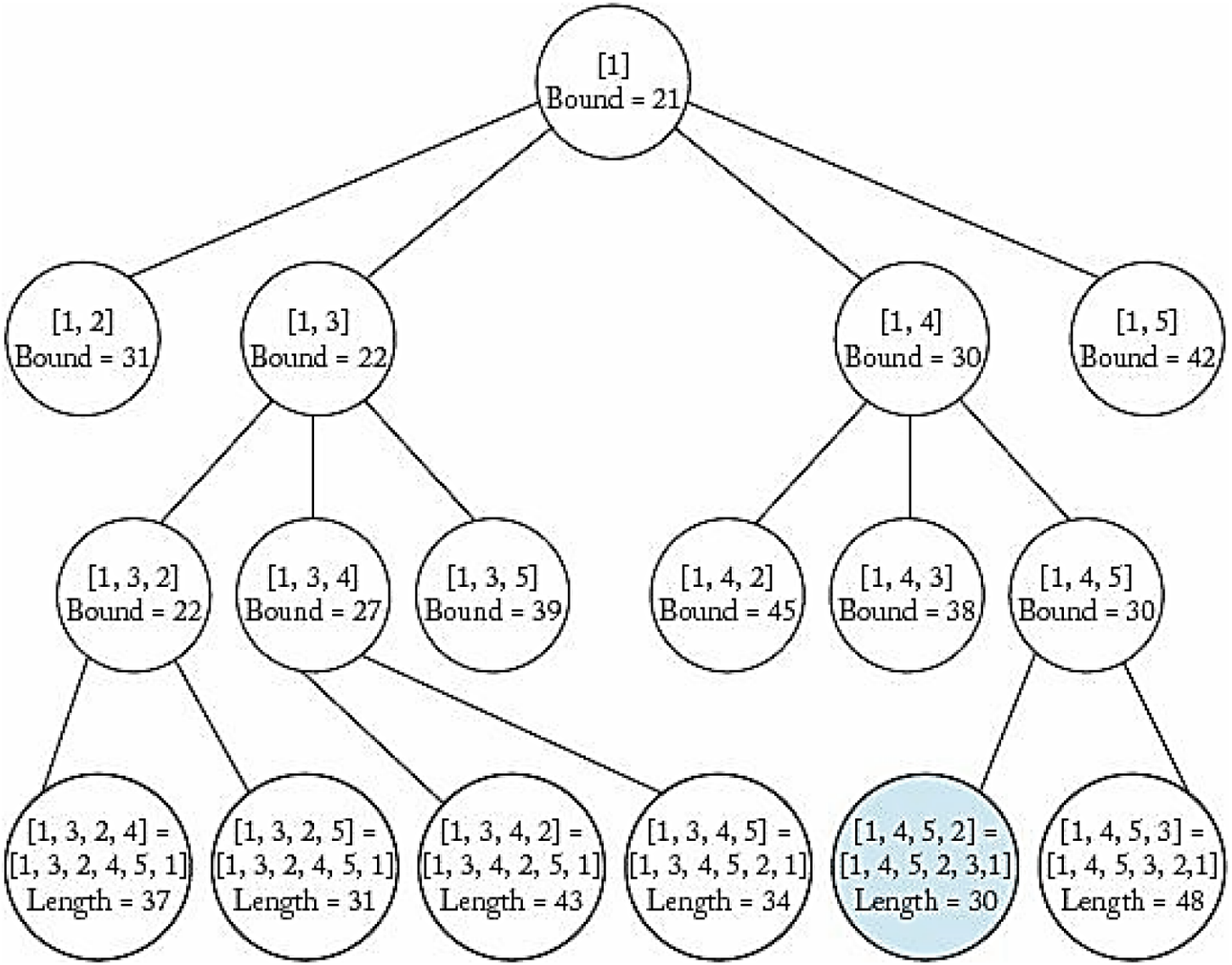
- **الگوریتم بهترین جستجو با هرس کردن شاخه و حد برای مساله فروشنده دوره گرد**

الگوریتم بهترین جستجو با هرس کردن شاخه و حد در حالت کلی

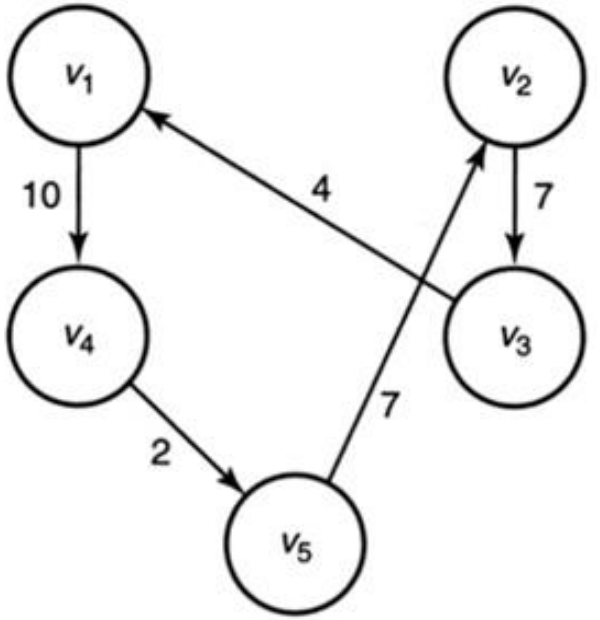
```
void best_first_branch_and_bound ( state_space_tree T, number& best)
 {
 priority_queue_of_node PQ;
 node u, v;

 initialize (PQ);                    // Initialize PQ to be empty.
 v = root of T;
 best = value(v);
 insert (PQ, v);

 while (! empty (PQ)) {              // Remove node with best bound.
    remove (PQ, v);
    if (bound (v) is better than best)       // Check if node is still promising.
        for (each child u of v) {
            if (value(u) is better than best)
                best = value (u);
            if (bound (u) is better than best)
                insert (PQ, u);
         }
  }
}
```



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



- تعیین تابع حد، منحصر به فرد نیست.
- مثلا در مساله فروشنده دوره گرد، می توان مینیم مقادیر هر ستون از ماتریس مجاورت را با هم جمع کرد (به جای مینیمم مقادیر موجود در سطرها).

- با توجه به اینکه هر راس باید دقیقا یکبار به عنوان ورودی و خروجی محسوب شود، هم از ستون ها و هم از سطرها می توان بهره برد. در این صورت، هزینه ملاقات هر راس برابر حاصل جمع وزن های مرتبط با ورود و خروج از راس می شود.

$$\frac{minimum\,(14,5,7,7)+minimum\,(14,7,8,7)}{2}=6.$$

پایان فصل ششم