طراحی و تحلیل الگوریتم ها

مدرس: سعدون عزیزی

s.azizi@uok.ac.ir

گروه مهندسی کامپیوتر

نیم سال دوم ۹۷-۹۶

تکنیک‌های طراحی الگوریتم‌ها

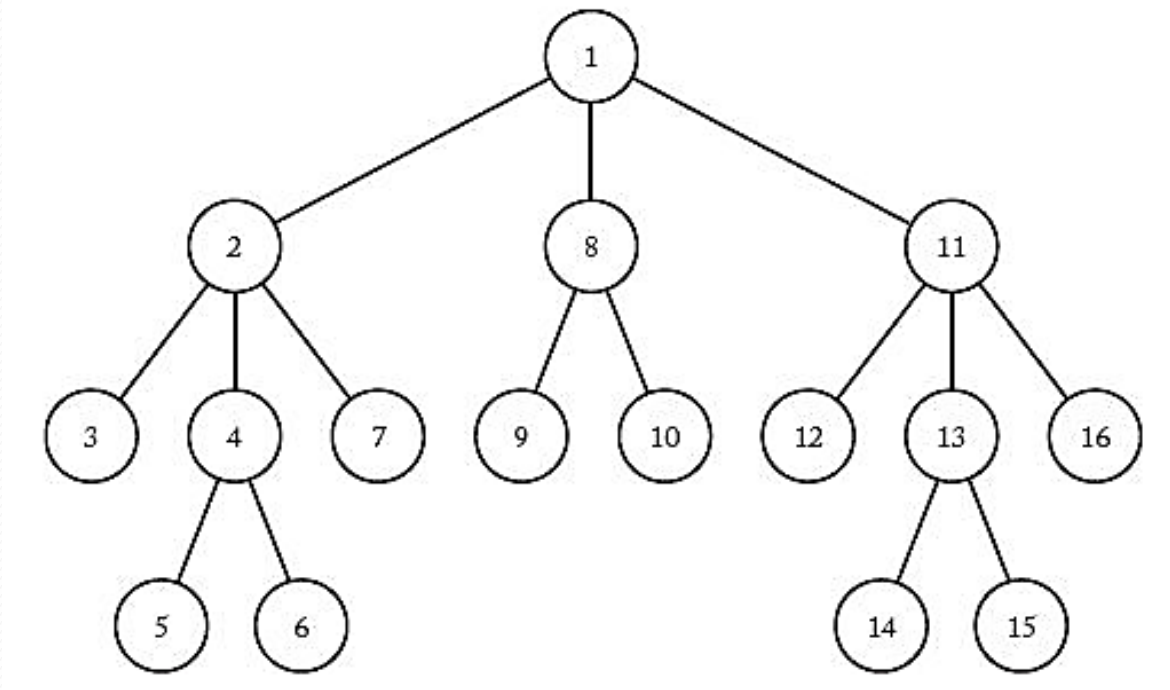
- تقسیم و حل
- برنامه‌ریزی پویا
- الگوریتم‌های حریصانه
- **عقبگرد**
- شاخه و حد

الگوریتم‌های عقبگرد (Backtracking Algorithms)

- در حالت کلی، از تکنیک عقبگرد برای حل مسائلی استفاده می‌شود که در آنها جواب به صورت دنباله‌ای از اشیاء (یک بردار) است: $s=(a_1,a_2,\ldots,a_n)$
- در هر گام، سعی می‌شود که یک جواب جزئی بدست آمده تا آن لحظه، $s_k=(a_1,a_2,\ldots,a_k)$، را با افزودن شیء بعدی به انتهای آن گسترش داد
- سپس بررسی می‌کنیم که آیا جواب جزئی بدست آمده یک جواب برای مسئله است یا خیر: اگر بود آن را گزارش می‌کنیم.
- اگر نبود، بررسی می‌کنیم که آیا جواب جزئی بدست آمده هنوز پتانسیل گسترش دارد یا خیر. اگر داشت ادامه می‌دهیم در غیر این صورت به یک مرحله قبل عقبگرد می‌کنیم.

الگوریتم‌های عقبگرد (Backtracking Algorithms)

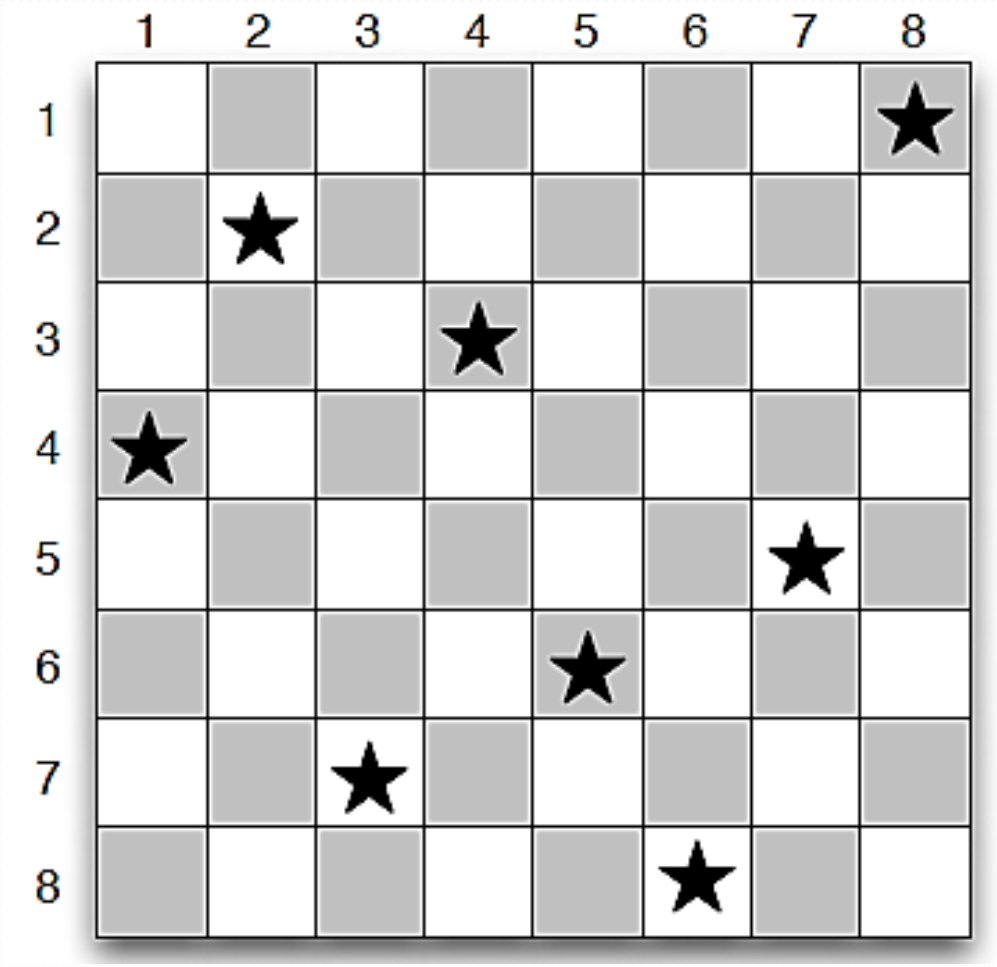
- الگوریتم‌های عقبگرد را می‌توان به عنوان جستجوی عمقی در یک درخت مدل کرد که در آن هر گره بیانگر یک جواب جزئی است.
- این درخت را درخت فضای حالت می‌گویند.



مسئله n–وزیر

- هدف از این مسئله، چیدن n مهره وزیر در یک صفحه شطرنج n×n است به طوری که هیچ دو مهره‌ای یکدیگر را تهدید نکنند.
- به عبارت دیگر، هیچ دو مهره‌ای نباید در یک سطر، ستون یا قطر یکسان باشند.
- در این مسئله، جواب به صورت دنباله‌ای از n موقعیت است که در آنها وزیرها قرار داده می‌شوند.
- مسئله n-وزیر، شکل کلی نمونه‌ای است از یک صفحه شطرنج استاندارد (n=8)

مسئله n–وزیر



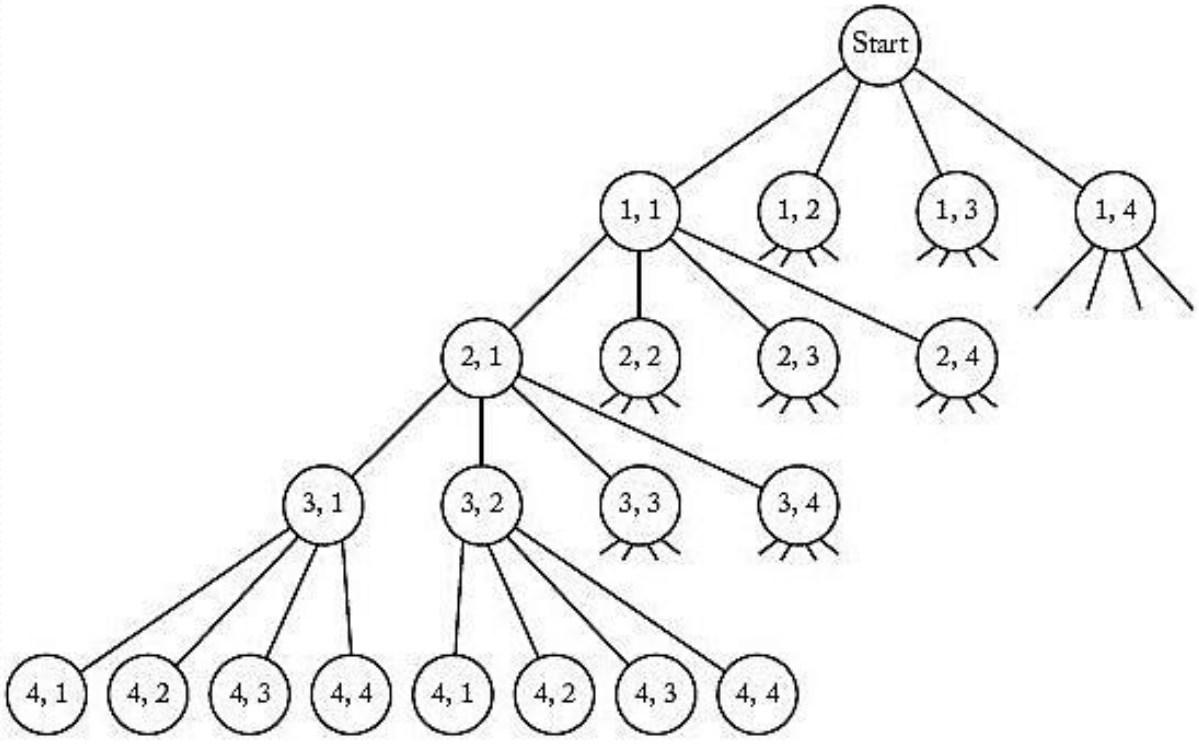
مسئله n–وزیر

روش‌های مختلف حل مسئله:

- جستجو در کل فضای جواب: $\binom{n^2}{n}$
- استفاده از هشت حلقه‌ی تودرتو: n^n
- استفاده از یک آرایه یک بعدی برای کاهش فضای جستجو

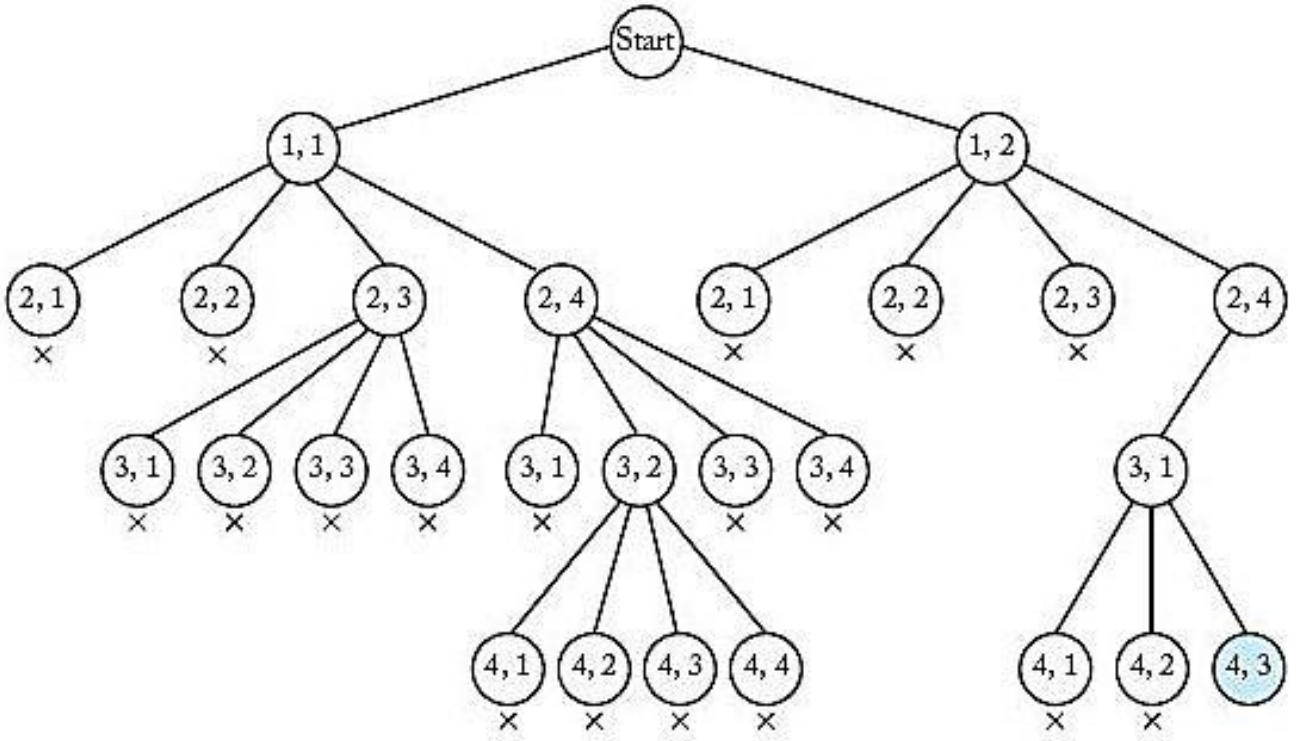
مسئله n–وزیر

- بخشی از درخت فضای حالت برای نمونه‌ای از مسئله n-وزیر (n=4):

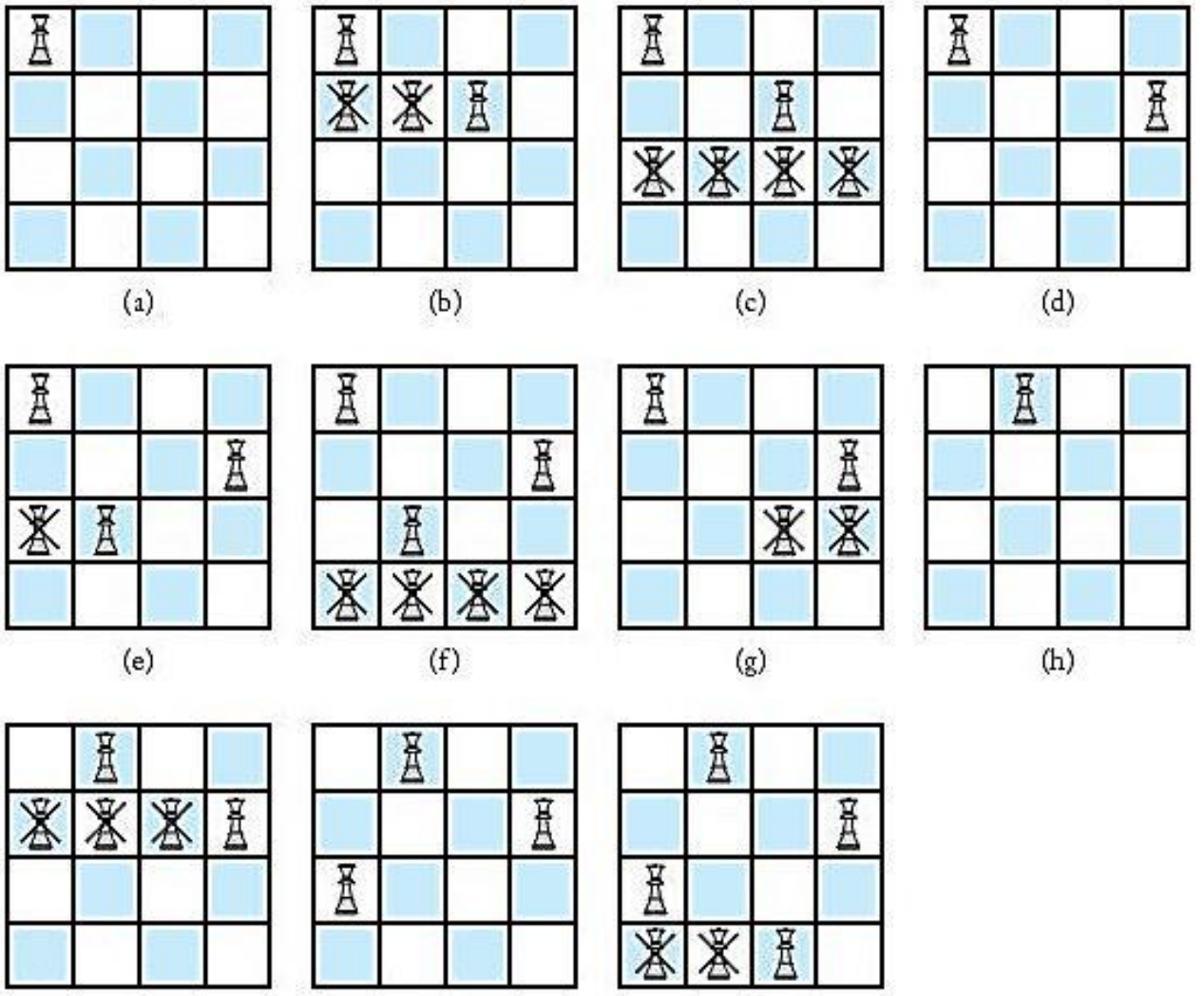


مسئله n–وزیر

- بخشی از درخت فضای حالت هرس شده (حذف حالت‌های تهدید در هر مرحله)
  - گره‌های غیرامیدبخش با علامت × مشخص شده‌اند



مسئله n–وزیر



مسئله n–وزیر

```
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);
}
```

مسئله n–وزیر

- تابع امیدبخش باید چک کند که آیا دو وزیر یکدیگر را تهدید می‌کنند یا خیر
- فرض کنید col(i)، i=1,2,..,n، ستونی باشد که در آن وزیر ردیف *i*اُم قرار می‌گیرد:
  - برخورد سطری قطعاً نخواهیم داشت (چون به ازای هر سطر یک وزیر خواهیم داشت)
  - تشخیص برخورد ستونی توسط رابطه زیر:

$$clo(i)=col(k)$$

  - تشخیص برخورد قطری توسط رابطه زیر:

$$clo(i)-col(k)=k-i \text{ یا } clo(i)-col(k)=i-k$$

مسئله n–وزیر

- تشخیص برخورد قطری:

قطر اصلی

	1	2	3	4	5	6	7	8
1	0	-1	-2	-3	-4	-5	-6	-7
2	1	0	-1	-2	-3	-4	-5	-6
3	2	1	0	-1	-2	-3	-4	-5
4	3	2	1	0	-1	-2	-3	-4
5	4	3	2	1	0	-1	-2	-3
6	5	4	3	2	1	0	-1	-2
7	6	5	4	3	2	1	0	-1
8	7	6	5	4	3	2	1	0

قطر فرعی

	1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8	9
2	3	4	5	6	7	8	9	10
3	4	5	6	7	8	9	10	11
4	5	6	7	8	9	10	11	12
5	6	7	8	9	10	11	12	13
6	7	8	9	10	11	12	13	14
7	8	9	10	11	12	13	14	15
8	9	10	11	12	13	14	15	16

مسئله n–وزیر (الگوریتم)

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

  if (promising(i))
     if (i == n)
        cout << col[1] through col [n];
     else
        for (j = 1; j <= n; j++){        // See if queen in
           col[i + 1] = j;               // (i + 1)st row can be
           queens(i + 1);                // positioned in each of
                                         // the n columns.
        }
}
```

مسئله n–وزیر (الگوریتم)

```
bool promising (index i)
{
    index k;
    bool switch;

    k = 1;
    switch = true;                    // Check if any queen threatens
    while (k < i &&  switch){         // queen in the ith row.
        if (col[i] == col[k] || abs(col[i] - col[k]) == i - k)
            switch = false;
        k++;
    }
    return switch;
}
```

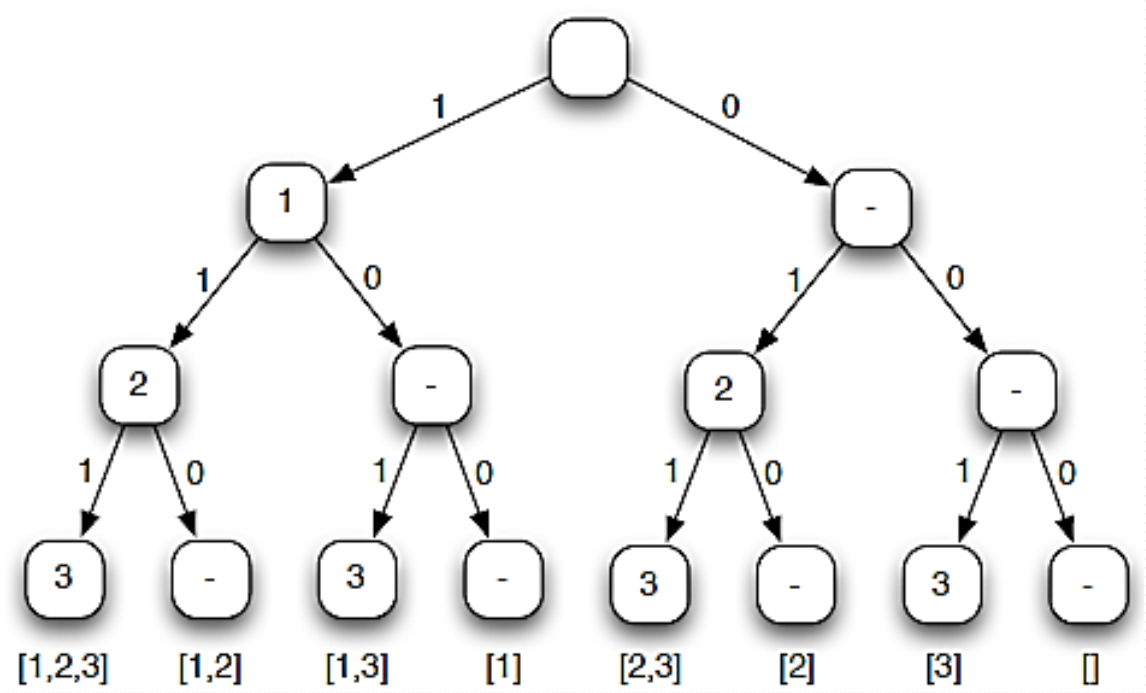
مسئله کوله‌پشتی ۰–۱ (مرور)

- مسئله کوله‌پشتی ۰-۱ (0-1 knapsack problem): یک دزد که در حال دزدی از یک مغازه است، n فقره جنس پیدا می‌کند؛ قیمت جنس iام v_i دلار و وزن آن w_i پوند است، که در آن v_i و w_i اعداد صحیح هستند. او می‌خواهد با ارزش‌ترین بار ممکن را بردارد، ولی کوله‌پشتی فقط به اندازه W پوند ظرفیت دارد که W یک عدد صحیح است. او باید کدام اجناس را بردارد؟ (از این رو به این مسئله، کوله‌پشتی ۰–۱ گفته می‌شود که در آن هر جنس یا برداشته می‌شود و یا خیر؛ به عبارت دیگر، دزد نمی‌تواند فقط قسمتی از یک جنس را بردارد. اجناس را به صورت شمش‌های طلا فرض کنید)

- مسئله کوله‌پشتی کسری (fractional knapsack problem): صورت مسئله همانند مسئله فوق است، با این تفاوت که دزد می‌تواند کسری از اجناس را بردارد و مجبور نیست برای هر جنس یک انتخاب دودویی صفر و یک انجام دهد. (در این حالت، اجناس را به صورت گرد طلا فرض کنید)

مسئله کوله‌پشتی ۰–۱

- می‌توان این مسئله را با استفاده از یک درخت فضای حالت حل کرد
- در هر مرحله، اگر به طرف چپ برویم قطعه مربوطه را برمی‌داریم، و اگر به سمت راست برویم آن قطعه را برنمی‌داریم.



مسئله کوله‌پشتی ۰–۱ (الگوریتم عقبگرد)

```
Backtrack-Knapsack(X, optX, optP, ℓ){
    if ℓ = n+1 then{
        if Σ_{i=1}^{n} x_i w_i ≤ b then{
            curP ← Σ_{i=1}^{n} x_i p_i;
            if curP ≥ optP then{
                optP ← curP;
                optX ← [x_1, x_2, ··· , x_n];
            }
        }
    }
    else{
        x_ℓ ← 1;
        Backtrack-Knapsack(X, optX, optP, ℓ+1);
        x_ℓ ← 0;
        Backtrack-Knapsack(X, optX, optP, ℓ+1);
    }
}
```

b: ظرفیت کوله‌پشتی

n: تعداد کل قطعه‌ها

X: جواب کاندید

ℓ: سطح درخت

curP: سود جواب کاندید

optX: جواب بهینه بدست آمده تاکنون

optP: سود بهینه بدست آمده تاکنون

تکنیک‌های طراحی الگوریتم‌ها

- تقسیم و حل
- برنامه‌ریزی پویا
- الگوریتم‌های حریصانه
- عقبگرد
- شاخه و حد

الگوریتم‌ها شاخه و حد (Branch and Bound Algorithms)

- این الگوریتم‌ها شکل بهبودیافته‌ای از الگوریتم‌های عقبگرد هستند
- هدف اصلی تکنیک شاخه و حد، افزایش سرعت الگوریتم‌های عقبگرد با استفاده از کاهش فضای جستجو می‌باشد
- الگوریتم‌های شاخه و حد در هر گره از درخت فضای حالت، یک کران (bound) را محاسبه می‌کنند تا تعیین شود که آیا آن گره امیدبخش هست یا خیر.
- اگر گره امیدبخش باشد مسیر مربوط به آن شاخه ادامه داده می‌شود وگرنه کار آن شاخه به پایان می‌رسد

مسئله کوله‌پشتی ۰–۱ (الگوریتم شاخه و حد ۱)

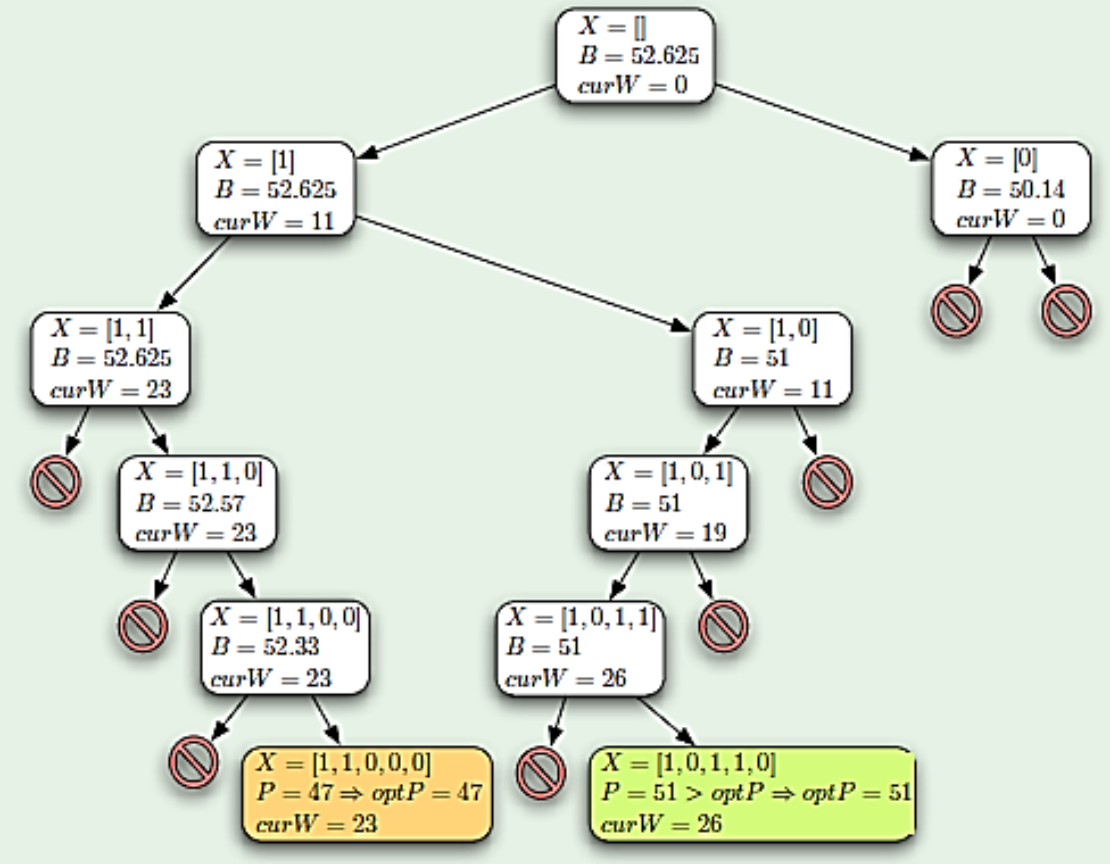
```
B&B-Knapsack1(X, optX, optP, ℓ, curW){
    if ℓ = n+1 then{
        if Σ_{i=1}^{n} x_i p_i ≥ optP then{
            optP ← Σ_{i=1}^{n} x_i p_i;
            optX ← [x_1, x_2, ··· , x_n];
        }
    }
    else{
        if curW + w_ℓ ≤ b then C_ℓ ← {1,0};
        else C_ℓ ← {0};
    }
    for each x ∈ C_ℓ do {
        x_l ← x;
        Backtrack-Knapsack(X, optX, optP, ℓ+1, curW + x_ℓ w_ℓ);
    }
}
```

مسئله کوله‌پشتی ۰–۱ (الگوریتم شاخه و حد ۲)

```
B&B-Knapsack2(X, optX, optP, ℓ, curW){
    if ℓ = n+1 then{
        if Σ_{i=1}^{n} x_i p_i ≥ optP then{
            optP ← Σ_{i=1}^{n} x_i p_i;
            optX ← [x_1, x_2, ··· , x_n];
        }
    }
    else{
        if curW + w_ℓ ≤ b then C_ℓ ← {1,0};
        else C_ℓ ← {0};
    }
    B ← Σ_{i=1}^{l-1} x_i p_i + GFK(p_ℓ, p_{ℓ+1}, ··· , p_n, w_ℓ, w_{ℓ+1}, ··· , w_n, b − curW);
    if B ≤ optP then return;
    for each x ∈ C_ℓ do {
        x_l ← x;
        Backtrack-Knapsack(X, optX, optP, ℓ+1, curW + x_ℓ w_ℓ);
    }
}
```

مسئله کوله‌پشتی ۰–۱ (الگوریتم شاخه و حد ۲)

مثال: فرض کنید که W={11,12,8,7,9}، P={23,24,15,13,16} و b=26 باشند. نتیجه اجرای الگوریتم شاخه و حد ۲ به صورت زیر است:



مسئله کوله‌پشتی ۰–۱ (مقایسه الگوریتم‌ها)

جدول زیر بدترین حالت اندازه فضای جستجو نمونه‌های تصادفی برای ۵ حالت مختلف را تحت الگوریتم‌های ارائه داده شده نشان می‌دهد.

n	Backtrack-Knapsack	B&B-Knapsack1	B&B-Knapsack2
8	511	333	78
12	8191	4988	195
16	131071	78716	601
20	2097151	1257745	480
24	33554431	19814875	755