بسم الله الرحمن الرحیم

جزوه طراحی الگوریتم

استاد:**مهندس محمد علی بحرینی**

E_Mail:malibahraini@gmail.com

منابع:

- **تحلیل و طراحی الگوریتم ها(مهندس جعفر تنها و دکتر احمد فراهی)**
- **جزوات الکترونیکی سایت**

فهرست

فصل اول:روش های تحلیل الگوریتم

- تحلیل الگوریتم ها
- پیچیدگی زمانی الگوریتم ها
- الگوریتم های بازگشتی(ترتیبی)
- طراحی الگوریتم ها

فصل دوم:روش تقسیم و حل(Divide and conquer)

- دیدگاه جزیی نگری در شناخت (reductionism)
- دیدگاه کل نگر در شناخت (holistic)
- تابع جستجوی دودویی به صورت بازگشتی(Binary search) وپیچیدگی زمانی آن
- تابع جستجوی دودویی به صورت غیر بازگشتی(Binary search) وپیچیدگی زمانی آن
- مرتب سازی ادغامی (Merge sort) وپیچیدگی زمانی آن
- مرتب سازی سریع(Quick sort) وپیچیدگی زمانی آن
- ضرب ماتریس ها به روش استراسن(Strassen) وپیچیدگی زمانی آن

فصل سوم:روش حریصانه(Greedy)

- الگوریتم حریصانه
- به دست آوردن درخت پوشای مینیمال (کمینه)
- الگوریتم پریم و پیچیدگی زمانی آن
- الگوریتم کراسکال و پیچیدگی زمانی آن
- الگوریتم دیکسترا و پیچیدگی زمانی آن
- مسئله ی خرد کردن پول
- مسئله ی کوله پشتی و تحلیل پیچیدگی زمانی آن
- ۳ نگرش برای حل حریصانه
- مسئله ی زمانبندی
- کد هافمن

فصل چهارم:برنامه نویسی پویا

- الگوریتم برنامه نویسی پویا
- ضریب دو جمله ای و تحلیل آن
- دنباله ی فیبوناچی و تحلیل آن

فصل پنجم:تکنیک عقب گرد(Backtracking)

- تکنیک عقب گرد
- تعریف حالت
- درخت فضای حالت
- بررسی امید بخش نبودن در مسئله ی ۴ وزیر
- الگوریتم امید بخش بودن برای مسئله n وزیر
- الگوریتم عقب گرد برای مسئله n وزیر

- الگوریتم مسئله حاصل جمع زیر مجموعه ها
- بررسی امید بخش بودن یک گره
- کاربرد رنگ آمیزی نقشه
- رنگ آمیزی گراف با ۳ رنگ
- مسئله ی دور همیلتونی
- مسئله ی کوله پشتی صفر و یک

فصل ششم:انشعاب و تحدید(Branch &Bound)

- ویژگی های کلی الگوریتم انشعاب و تحدید
- کوله پشتی صفر و یک
- فروشنده ی دوره گرد
- محاسبه هزینه تور

فصل هفتم:مسائل رام نشدنی

- مسائل تصمیم گیری و دسته بندی آن ها
- کلاس p
- کلاسnp
- Np complete
- Np hard

فصل اول

روش های تحلیل الگوریتم

تعریف الگوریتم : یک سری راه حل و دستورات که دارای شروع و پایان است و برای رسیدن به هدف خاص و به منظور حل مسئله طراحی می شود

شرایط الگوریتم : ۱- دقیق ۲- انجام مراحل به ترتیب ۳- پایان پذیر

مشخصه های دیگر الگوریتم :

۱- زمان اجرا ۲- تعداد حافظه های مصرفی ۳- کارایی دریچه نو contribution : در رابطه با صفحه بعد آن قسمتی که در رابطه با ساختار اطلاعاتی و معرفی آن است.

طراحی الگوریتمها : شامل روش – شرح – کلی – ساختار اطلاعات

۱- روش تقسیم و حل : تقسیم مسئله به دو یا چند زیر مسئله تا جایی که هر مسئله به تنها یی قابل حل باشد – پشته

۲- روش حریصانه : حل مسئله به صورت مرحله ای به طوری که در مرحله با انتخاب یک ورودی یک گام به جواب نزدیک تر می شویم – آرایه

۳- روش برنامه نویسی پویا : حل مسئله از پایین به بالا و یافتن یک رابطه بین حل مسئله با اندازه های بالاتر و اندازه های پایین تر (مثال فیبوناچی – ارایه)

مثال : $f(n) = f(n-1) + f(n-2)$

۴- تکنیک عقبگرد : نحوه ای از پیمایش درخت مسئله به صورت عمقی عقبگرد از گره ی فعلی در صورت امید بخش نبودن (درخت و پشته)

۵- روش انشعاب و تحدید : نحوه ای از پیمایش درخت مسئله به صورت عرضی و رفتن به گره سطح بعدی در صورت امید بخش نبودن گره فعلی (درخت و صف)

۶- الگوریتم ژنتیک : حل مسئله در زمان واقعی با روشهای تکاملی با الهام از طبیعت (تئوری کامل داروین) – آرایه

عوامل دخیل در اجرای برنامه :

۱- سرعت نرم افزار
۲- نوع کامپایلر

} اثر ثابت در زمان اجرا

۳- اندازه داداه ی ورودی ⟵ n

مثل quik sort یک عنصر در نظر میگیریم هر آنچه کوچکتر بود یک طرف آن و هر چه از آن بزرگتر بود به طرف دیگر می فرستیم مسلما برای یک آرایه مرتب ، بدترین حالت و برای

آرایه غیر مرتب ، بهترین حالت و برای حالت متوسط ، ترکیبهای مختلف داده ها را در نظر میگیریم.

۴-ترکیب داده های ورودی : بسته به شرایط (از بدترین حالت – متوسط - بهترین حالت)

۵- پیچیدگی زمانی الگوریتم: تابعی از اندازه های مسئله

۶- پارامترهای دارای تاثیر ثابت در زمان اجرا

N اندازه ورودی مسئله	T(n) تابع زمانی الگوریتم
چند جمله ای	$T(n) = 2n^2 = n$
نمایی	$T(n)= e^n$
نمایی	$f(n) = 2^n + \log n$
لگاریتمی	$T(n) = \log n$

Road map : نقشه کلی درس طراحی الگوریتم ها

تحلیل الگوریتمها : بررسی زمان پیچیدگی

الگوریتمهای ترتیبی

الگوریتم های بازگشتی

تحلیل

حل

❊ توجه شود که ۳ فصل اول طراحی الگوریتم ها مربوط به فصل اول و آخر ساختمان داده ها است.

نکته مهم : از مبانی کامپیوتر درس استاد، ساختار اطلاعاتی مسئله روی طراحی الگوریتم ها تاثیر گذار است.

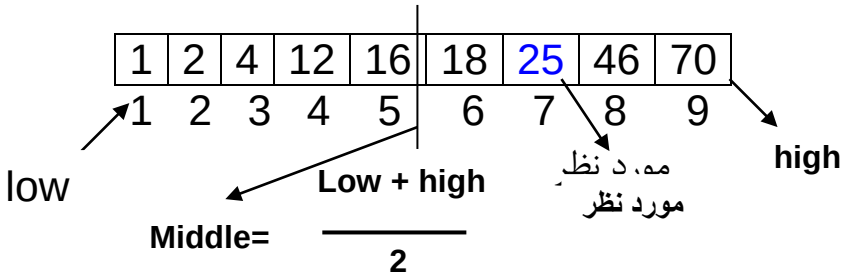
نماد و معماری کلی در واقع همان طراحی الگوریتم است. در ابتدا باید نقشه کلی را داشته باشیم تا در نهایت دچار سر درگمی نشویم.

- هر چیزی در دنیا یک سیستم است و دارای ورودی و خروجی و پردازش است

مراحل تبدیل از ورودی به خروجی طراحی الگوریتم ها می باشد.

مثالهای برای روشهای حل الگوریتم

۱- طراحی الگوریتم جستجوی دودویی binary search



تقسیم مسئله : به دو زیر مسئله و یافتن جواب در هر یک از آنها

```
Bin search ( low + high , x )
{
If  ( x < middle )
Bin search ( low , middle )
Else
Bin search ( middle + 1 , high ) ; }
```

هسته core

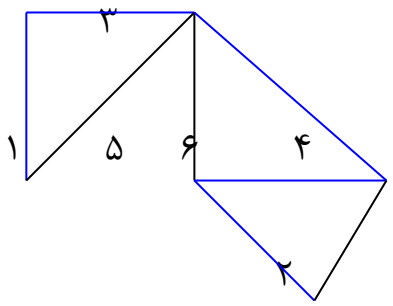
شبه کد بیان الگوریتم به زبان c بازگشتی و فراخوانی تابع pseudo code

یادآوری : برای توابع بازگشتی از پشته استفاده میشود

مزیت این روش: سادگی کد و طراحی

ایراد: زمان بالا و پر کردن و خالی کردن پشته برای رسیدن به جواب نهایی

روش ۲ مثال : الگوریتم پریم در ساختمان داده ها برای یافتن درخت پوشای مینیمال در یک گراف (یعنی نباید حلقه داشته باشیم)

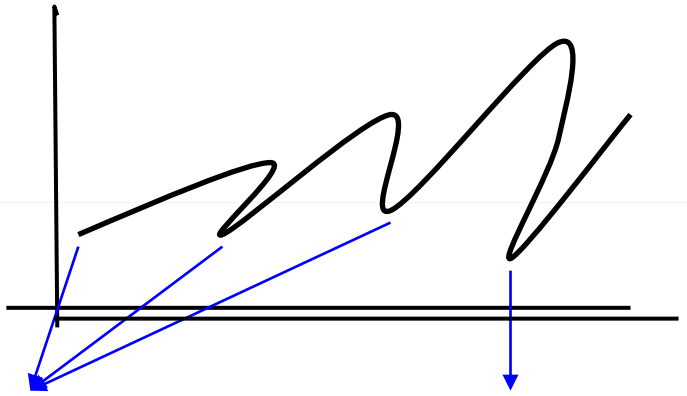


sum = 3 + 1 +4 + 2 + 1 = 11

۱ ۳

تمام رئوس پوشش داده میشودو مجموع وزن یالها مینیمم است نقطه شروع روی جواب تاثیر می گذارد.

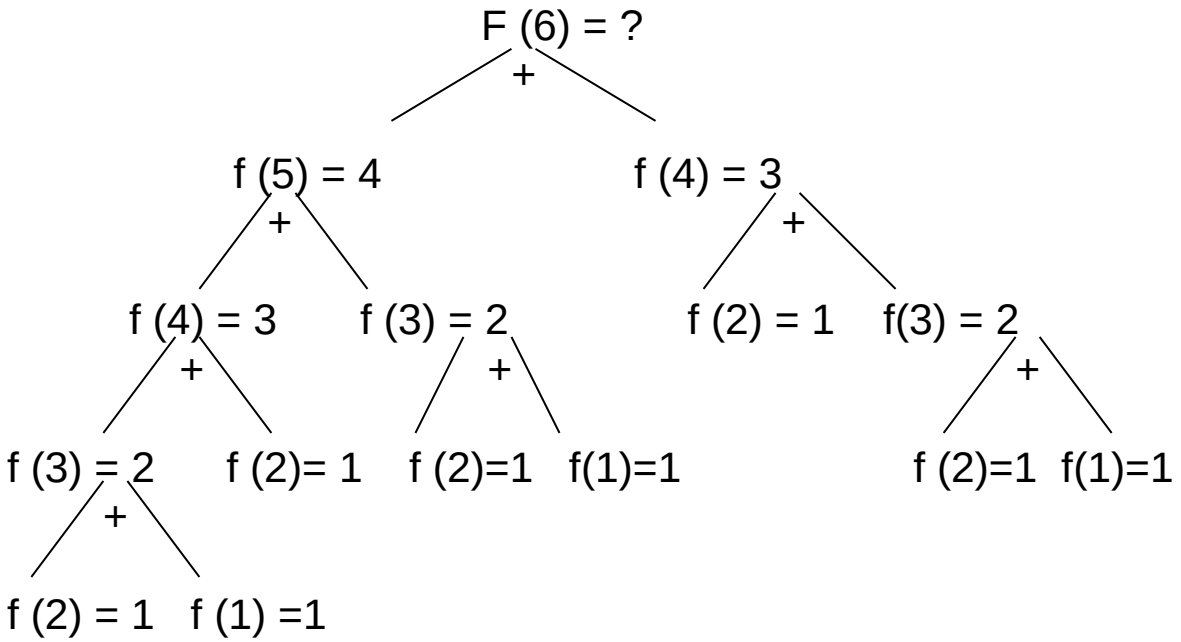
حل مسئله به صورت مرحله ای – در هر مرحله یک ورودی و یک گام به جواب نزدیک می شویم.



Min محلی min مطلق

روش اول : درخت _ مسئله به صورت بازگشتی (فیبونانچی)

F (n) = f (n-1) + f (n-2)
F (2) = f (1) = 1



ایراد : محاسبه یک سری از موارد تکراری مثل f (4) – f (3)

روش ۳ : در برنامه نویسی پویا مقادیر مسئله از پایین به بالا و در یک آرایه نگهداری میشود در نتیجه مقادیر تکراری محاسبه نمی شود.

1	1	2	3	5	8	13....
1	2	3	4	5	6	7

از پایین به بالا به کمک ساختار اطلاعاتی آرایه و یک حلقه for ساده

روش چهارم : مثال – مسئله ۴ وزیر یک ماتریس ۴*۴ که میخواهیم ۴ وزیر را به طوری در آن قرار دهیم که یکدیگر را تحدید نکنند.

*			
		*	

این جواب نمی دهد از اینجا باید عقبگرد کنیم حتما باید در هر سطر یک وزیر باشد در نتیجه مسئله قابل حل نیست.

	*		
			*
*			
		*	

feasible

OR : operation pesearch : تحقیق در عملیات

مروری بر ساختمان داده ها:

تحلیل الگوریتمهای ترتیبی

مسئله ۱) برای تکه کدهای زیر تابع پیچیدگی زمانی را بدست آورید

```
1 )  x = 0
2)  ( i = 0 ; I < n ;  i++ )
3)  x++
```

اگر مساوی داشت سطر ۲ و ۳ تغییر نمی یافت و هر کدام یکی اضافه میشد

به خاطر این n + 1 شد اگر I++ بود ، n میشد زیرا i++ یعنی بعد از اضافه شدن همواره یک بار بیشتر اجرا می شد

سطر	زمان	تعداد	جمع
۱	C_1	1	C_1*1

2	C_2	N+1	$C_2(n+1)$
3	C_3	n	$C_3 (n)$

جمع کل : $t (n) = C_1 + C_2(n+1) + C_3 n$

$C = \max (C_1 , C_2 , C_3) \longrightarrow t (n) = c (2n + 2)$

مسئله ۲) باید توجه شود که اگر for داشته باشیم بزرگترین زمان اجرای درون آن را در نظر میگیریم یا مجموع آنرا

```
1)  x = 0
2)  for ( i=0 ; i< n ; i++ )
3)      for ( j=0 ; j<n ; j++ )
4)  C1          X++ ;
```

سطر	هزینه واحد	تعداد	هزینه کل واحد
1	C_1	1	C_1
2	C_2	n +1	$C_2(n+1)$
3	C_3	n(n+1)	$C_3(n(n+1))$
4	C_4	n*n	$C_4(n*n)$

$T (n) = C_1 + C_2(n+1) + C_3(n(n+1)) + C_4(n*n)$

$C = \max (C_1, C_2 ,C_3 , C_4) \Longrightarrow c (2n^2 + 2n + 2)$

مسئله ۳) تابع فاکتوریل

توجه : درواقع در یک حلقه تودرتو دستورات درونی فقط تحت تاثیر قسمت اول حلقه for یعنی از ۰ تا n بار تکرار میشود و نه تحت تاثیر i++

```
1)  int factorial ( int )
    {
2)    Int fact = 1;
3)    For ( int i=2 ; i<=n ; i++ )
4)          Fact * i ;
          Return fact ; }
```

سطر	هزینه	تعداد	هزینه سطر
1	C_1	0	0
2	C_2	1	C_2
3	C_3	n	C_3n

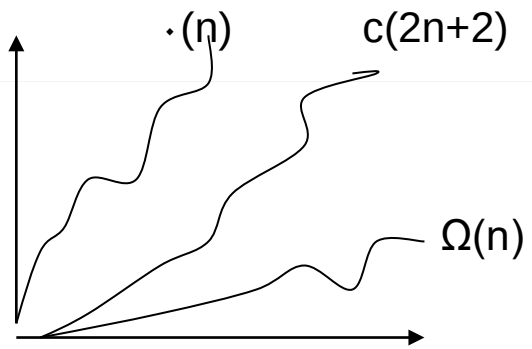
4	C_4	n-1	$C_4(n-1)$
5	C_5	1	C_5

$T(n) = C_2 + C_3n + C_4(n-1) + C_5$

$c = \max(C_1 \ldots\ldots C_5) = \max(C_i)$

$t(n) = C(2n+1)$

تمرین) در فایل excel مقادیر n از ۱ تا ۱۰۰ با نمودار کشیده شود



نماد Big – oh حد بالای t(n)

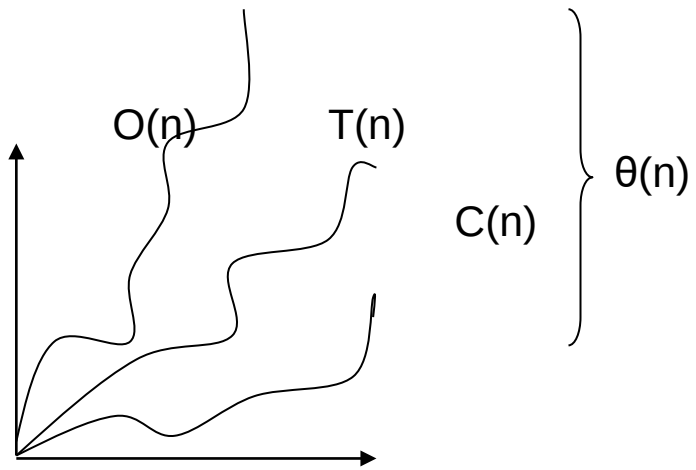
$T(n) = C(2n+2) \quad t(n) = c(2n+2) < 2\,Cn + 2\,Cn \quad T(n) \in o(n)$

نماد big – omega ، کران پایین برای زمان اجرا

$T(n) = C(2n+2) \quad T(n) > 2Cn > Cn \quad T(n)\ \Omega(n)$

نماد θ ، معرف حد چپ و پایین – حد میانگین

$Cn < T(n) < 4Cn$



روشهای تحلیل و حل روابط بازگشتی:

روشهای تحلیل : ۱) فراخوانی : ۱. انتقال متغیرهای محلی به پشته ۲. انتقال آدرس بازگشتی به پشته ۳. انتقال پارامترها ۴. انتقال کنترل به ابتدای پردازنده ی جدید

۲) بازگشتی : عکس عملیات فوق

روشهای حل روابط بازگشتی :

۱) روش استقرا یا حد : ۱. حدس جواب ۲. اثبات برای یک مقدار مشخص اولیه ۳. اثبات برای گذر از $k < n$ به n

۲) روش تکرار با جایگذاری : ۱. جایگذاری مکرر رابطه بازگشتی در خود فرمول تا رسیدن به وضعیت اولیه (فرمول کلی تابع بازگشتی) ۲. درخت بازگشتی : تشکیل درخت بازگشتی شامل : تعداد گره بازگشت به عنوان رشته – ایجاد گره به تعداد عملیات بازگشتی در سطح بعدی – تکرار مرحله قبل تا رسیدن به n مقدار ثابت ۳. محاسبه مقدار رابطه بازگشتی : محاسبه ی هر سطح درخت – محاسبه مجموع کلی عبارت دو سطح

۳) روش قضیه اصلی:

$T(n) = at(n/b) + f(n)$

$(a,b>1)$ رابطه کلی

G درجه رشد = $g(n) = n^{\log_a b}$

F درجه رشد = $F(n)$

الف) $F > G$: $t(n) \in (f(n))$

ب) $F < G$: $T(N) \in \theta(G(N))$

ج) $F = G$: $T(n) \in \theta(g(n)\log^n)$

۴)روش کلی ۱) رابطه بازگشتی مثل معادله دیفرانسیل مرتبه ۱

$T(n) = \{ b_n t(n) + c_n (n) > 0 \}$

$C(n=0)$

۱. همگن $C=0$

۲. غیر همگن $C \neq 0$ ⟵ تغییر متغیر

$T(n) = b_1 b_2 \ldots b_n A(n)$

$d_n = \dfrac{c_n}{b_1 b_2 \ldots b_n} \longrightarrow t(n) = (b_1 b_2 \ldots b_n)(T(0) + b_1 \ldots b_2)$

۲)رابطه بازگشتی مرتبه دوم:

$At(n-1) + bT(n-2) \ (n>1)$

$T(0) = C_0 + T(1) = C$ $\quad$ $T(n) \rightarrow$ تغییر متغیر $T(n) = x^n$

جایگذاری دو رابطه بازگشتی $\rightarrow$ $x^n = a\,x^{n-1} + b\,x^{n-2} \longrightarrow x^2 - ax - b$

$\begin{cases} x_1 \\ x_2 \end{cases}$ $\quad$ $\begin{cases} x_1 , x_2 \text{ متمایز} \\ x_1 , x_2 \end{cases}$ حل میکنیم

$\longrightarrow T(n) = c_1 x_1^2 + c_2 x_2^n$ متمایز: c_1 , c_2 را به کمک شرایط اولیه تعیین میکنیم

x_1 و x_2 یکی هستند $\longrightarrow T(n) = c_1 x_1^n + c_2 n x_2^n$

۳) رابطه مرتبه دوم غیر همگن

$$T(n) = a\underbrace{T(n-1)}_{Y'} + b\underbrace{T(n-2)}_{y''} + c\,F(n)$$

مانند معادلات دیفرانسیل مرتبه دوم غیر همگن $\longleftarrow$ تبدیل به همگن با حذف جمله غیر همگن

معادله معمولی $x^2 + x - 1 = 0$ $\longrightarrow$ x باید بدست بیاید

$Y - x^2 - yx = 0 \longrightarrow y(x)$: y بدست میاید x بر اساس

معادله دیفرانسیل که مشتقات y در آن وجود

$y' = x \longrightarrow y(x) \xrightarrow{?} y = x^2/2 + c$ انتگرال میگیریم

$y'' - 2y'x + y = x^2 \xrightarrow{y=?}$ معادله دیفرانسیل مرتبه ۲

ابتدا برابر صفر میگیریم همگن میکنیم و جواب همگن را بدست می آوریم.

فصل دوم

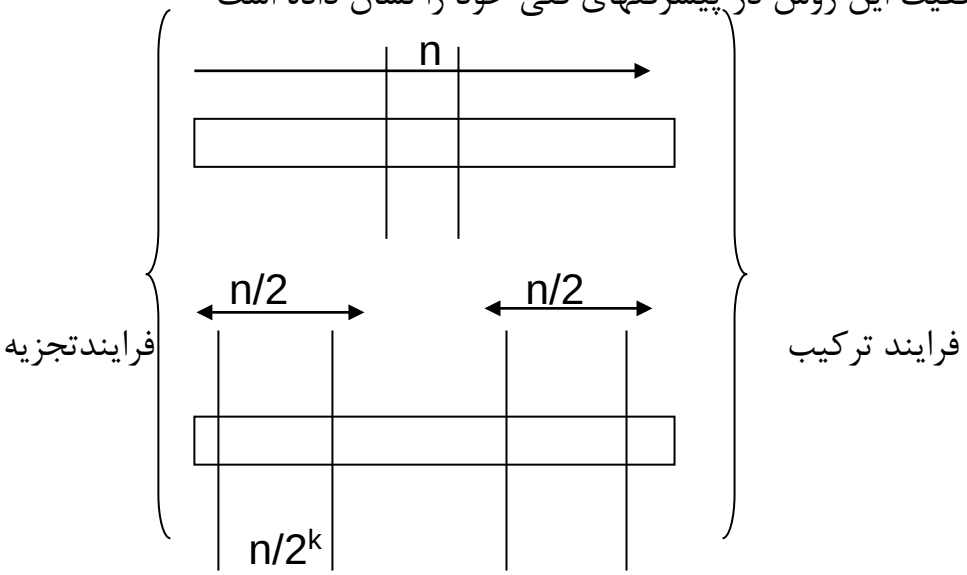
روش تقسیم و حل(Divide & conquer)

روش تقسیم و حل (divid & conquer):

دیدگاه جزیی نگری در شناخت (reductionism)

در دیدگاه دکارتی برای شناخت میتوان ماهیت مورد نظر را به دو نیمه تقسیم کرد و هر نیمه را شناخت . در این صورت با ترکیب در شناخت ، شناخت کامل حاصل میگردد که در صورت عدم شناخت دو نیمه هزینه را باز به دو نیمه دیگر تقسیم میکنیم و آنقدر ادامه میدهیم تا شناخت حاصل شود . این دیدگاه مبتنی بر فلسفه غربی شناخت دارد . منشا آن دیدگاه اتمی است . دیدگاه اتمی در شناخت به این نکته اشاره دارد ، کار در نهایت امر تقسیم به جزیی غیر قابل تقسیم منجر میگردد که باید آن را شناخت

- موفقیت این روش در پیشرفتهای فنی خود را نشان داده است



شکل – نگرش کلی به مسئله تقسیم و حل

دیدگاه کل نگر در شناخت (holistic):

در این دیدگاه عمدتا مبتنی بر نگرش شهرتی میباشد در ابتدا یک شناخت کلی از مسئله و سیستم حاکم بر ان حاصل میگردد و عمدتا به جای تجزیه و تحلیل سیستم بر حسب اجزای

ان برای شناخت و رفتارهای سیستم و اثرات متقابل ان با محیط و سایر سیستم ها مورد مطالعه قرار میگیرد

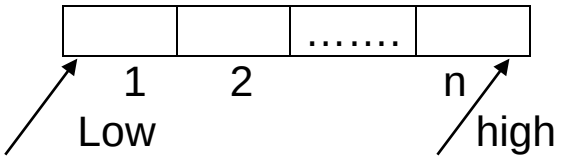
- موفقیت این دیدگاه عمدتا خود را در سیستمهای تولیدی ژاپن بعد از جنگ جهانی دوم و تهدید جدی شرکتهای خوروسازی غربی تا مرز ورشکستگی خود را نشان داده است (فلسفه شهرتی JIT)

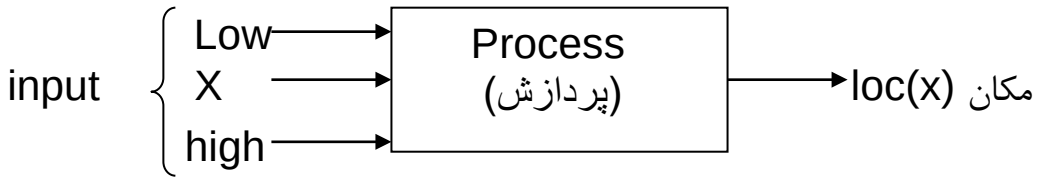
مسئله : برنامه ای بنویسید که در فهرستی از عناصر دریافتی که به صورت صعودی مرتب شده است عنصر X را در صورت وجود بیابد و مقدار انرا اعلام نماید

مراحل کار :

۱. شناخت و تجزیه تحلیل
۲. انتخاب یا طراحی ساختار اطلاعاتی
۳. طراحی الگوریتم و فلوچارت
۴. پیاده سازی
۵. خطا زدایی

مرحله ۱- **شکل 1)** شمای کلی (سیستمی) جستجوی دودویی





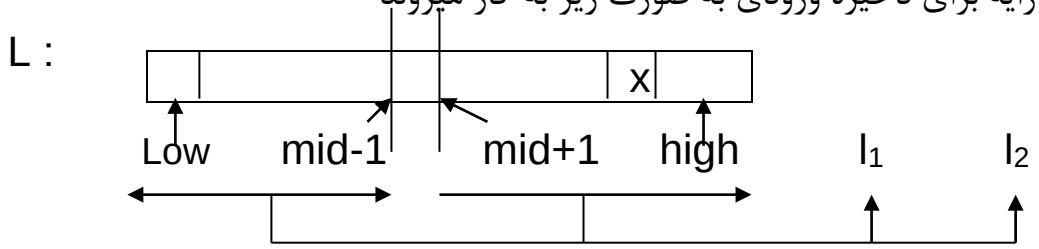
binsaerch (l , low , high , x) output

شکل ۲) شمای کلی الگوریتم

	شروع
ورودی	low ←ورودی high←ورودی x ← ورودی l ← ورودی
پردازش	loc ←binsearch (low,high,l,x)

خروجی	خروجی ← loc

مرحله ۲: آرایه برای ذخیره ورودی به صورت زیر به کار میروند



$$Mid = \frac{low + high}{2}$$

شکل ۳ – ساختار اطلاعاتی مسئله برای ذخیره ورودی

مرحله ۳ ایده کار با روش تقسیم و حل

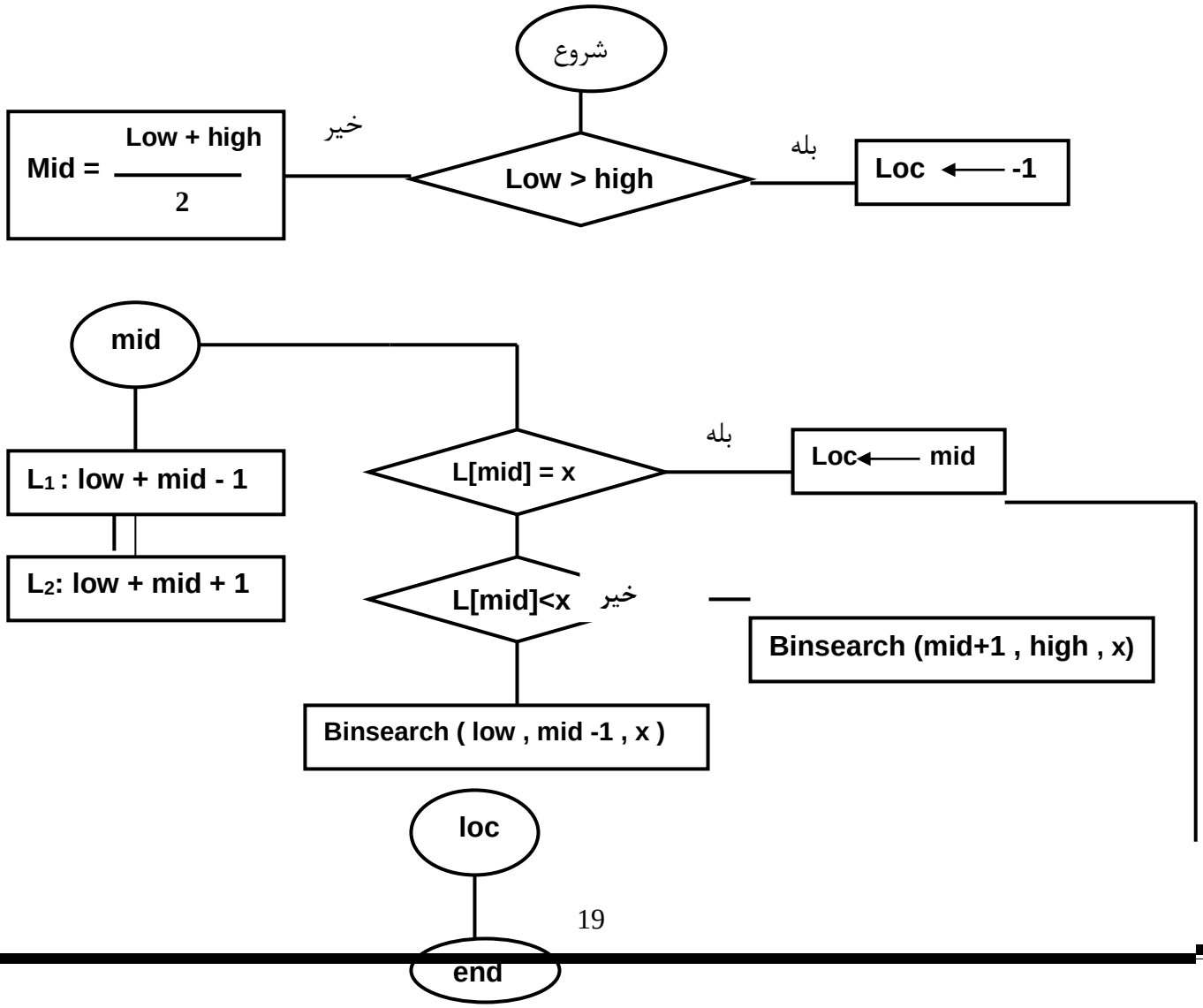
تقسیم ساختار اطلاعاتی به دو قسمت و یافتن مقدار x در یکی از آنها با کمک فراخوانی تابع جستجوبه صورت بازگشتی

- نکته : ساختار اطلاعاتی مسئله روی الگوریتم حل تاثیر میگذارد
- کشیدن تصویر درک بهتری نسبت به توصیف شفاهی آن در چند صفحه به ما می دهد.

ورودی	۱. دریافت آرایه ورودی مرتب ورودی ← i ورودی ← low ورودی ← high ورودی ← x
پردازش	۲. اگر low > high ← ۱- ← loc ۳. یافتن عنصر وسط آرایه $Mid = \frac{low + high}{2}$ ۴. تقسیم آرایه به دو نیمه با کمک عنصر وسط ۵. اگر عنصر وسط x = mid ← mid ← Loc وبرو به x ۶. اگر mid>x فراخوانی الگوریتم برای نیمه

	اول آرایه (فراخوانی بازگشتی) ۷. اگر mid <x , فراخوانی الگوریتم برای نیمه دوم آرایه (فراخوانی بازگشتی)
خروجی	۸. loc ⟵ خروجی
	۹. پایان

نکته : l و x پارامترهای ثابت در تمام فراخوانی بازگشتی هستند و لذا میتوان به عنوان پارامتر در پیاده سازی از انها به جهت استفاده مناسب از حافظه صرف نظر کرد در این صورت نیازی به ذخیره انهادر پشته نیست.



شکل ۵

شکل ۵ : نمای توسعه یافته فلوچارت جستجوی دودویی برای تابع search حافظه

مرحله ۳– پیاده سازی با شبه کد C

برنامه نویسی از بالا به پایین top down programming چرا که اگر بخواهیم تغییری اعمال کنیم مراحل قضیه فقط و فقط تابع تغییر میکند نه بدنه اصلی برنامه

```
# include<stdio.h>
# include<conio.h>
it  binsearch (l, low , high , x )
{ i ( low > high )
          Return ( -1 )
Else {
       Mid = low + high / 2;
         i x== l[mid]
           Return mid ; ──────► خروجی
         Else i ( x < a[mid]
                Return binsearch ( l,low,mid-1,x) ──────► T(n/2)
                 Else return binsearch 9 l,mid+1,high) ──────► T(n/2)
         }
```

این تمرین را در c یا c++ اجرا کرده و فایل اصلی و اجرایی آن را ارسال کنید

```
          Void main () {
                Cin >>1;
ورودی           Cin>>low;
                 Cin>>high;
                 Cin>> x;
```

پردازش loc = binsearch (l , low , high , x)

خروجی cout << loc

پیچیدگی زمانی مسئله قبل (جستجوی دودویی)

$$T(n) \begin{cases} C & n = 1 \\ T(n/2) + d \longrightarrow \text{یک سری خورده زمانهای اضافی که زمان ثابتی دارند} \end{cases}$$

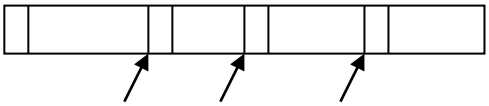
$$T(n/2) + d = T(n/2^2) + 2d = \ldots = T(n/2^k) + kd$$

$$n/2^k = 1 \longrightarrow k = \log^n{}_2$$

$$T(n) = c + d \log^n{}_2 = O(\log n) \qquad T(n) = \begin{cases} O(\log n) & \text{جستجوی موفق} \\ \Theta(\log n) & \text{جستجوی ناقص} \end{cases}$$

تابع جستجوی دودویی به صورت غیر بازگشتی:

روشهای بازگشتی به دلیل زمانبر بودن و مصرف نامناسب حافظه مناسب نیستند حتی الامکان به دنبال روشهایی هستیم که مستقیما به جواب منتهی شوندمیتوان از دیدگاه تقسیم و حل به نحوی استفاده کرد

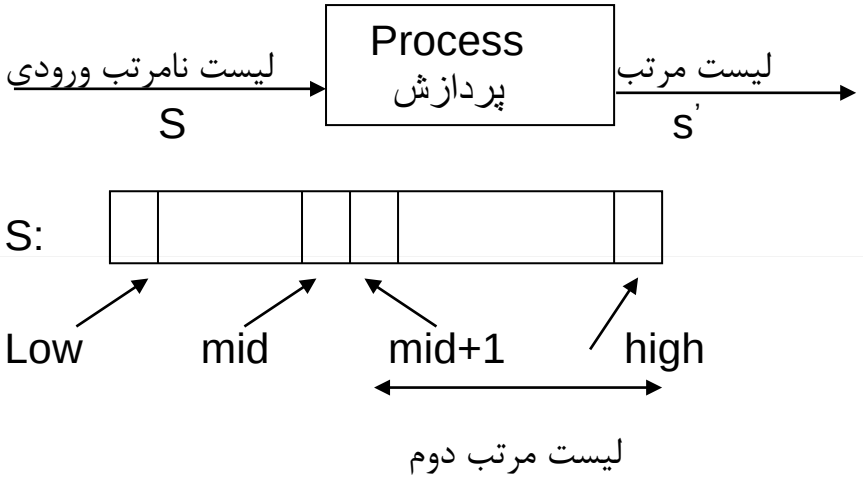


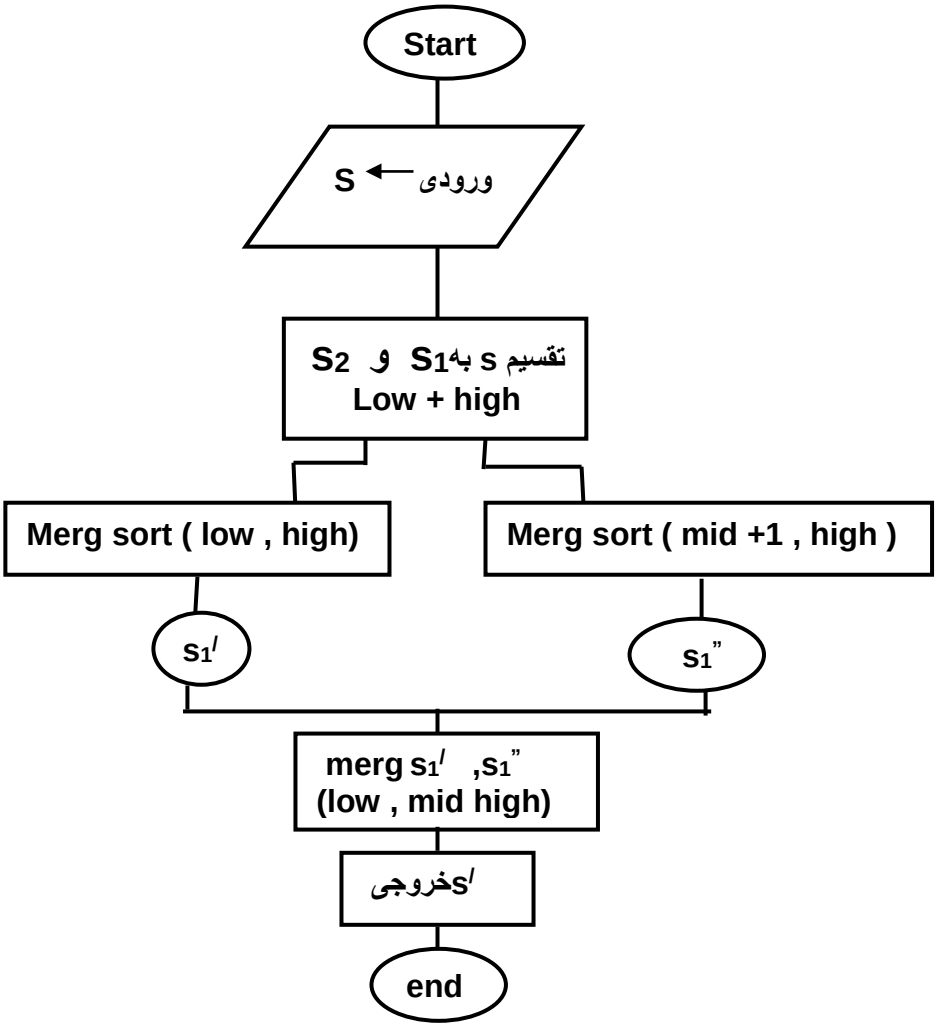
	شروع
ورودی	۱. دریافت ورودی (x , high , low , l)
پردازش	۲. mid ⟵ (low + high)/2
	۳. اگر x = l[mid] ⟹ mid ⟵ low برو به ۸
	۴. اگر x < l[mid] ⟹ m-1 ⟵ high
	۵. برو به ۲
	۶. اگر x > l[mid] ⟹ mid – 1 ⟵ high
	۷. برو به ۲ (اطلاعات قبلی مورد نیاز نیست لذا پشته ورودی ندارد)
خروجی	۸. loc ⟵ خروجی
	پایان

تمرین: این برنامه را در محیط برنامه نویسی بنویسید.

مرتب سازی ادغامی:

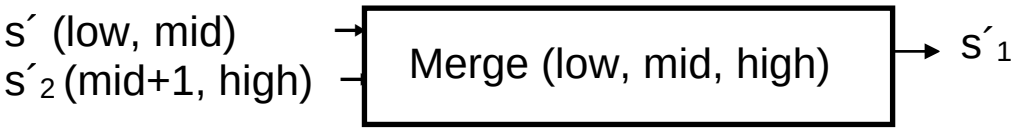
۱. تقسیم آرایه به دو زیر آرایه هر یک با n عنصر
۲. حل هر زیر آرایه با مرتب سازی آن
۳. ترکیب حل های زیر آرایه از طریق ادغام انها در یک آرایه مرتب





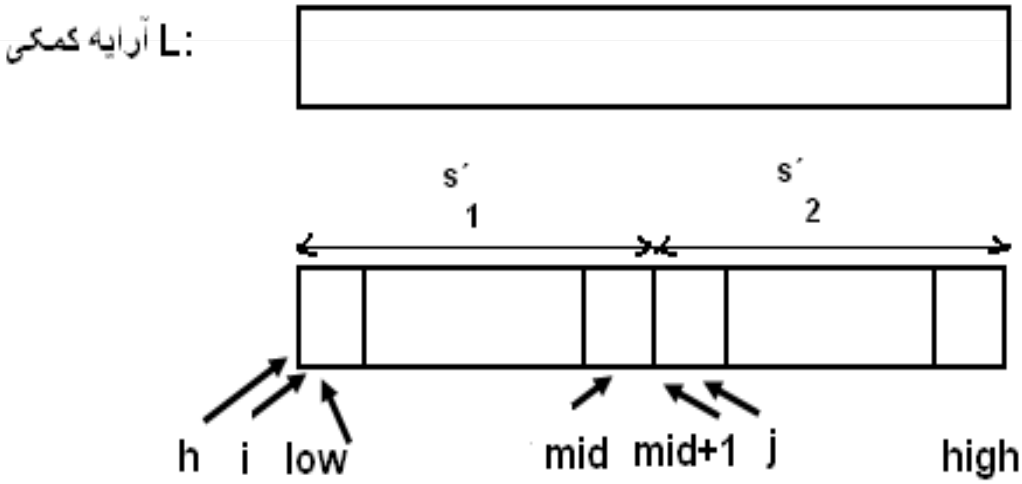
الگوریتم ادغام دو آرایه ی مرتب و به دست آوردن یک آرایه ی مرتب:

✓ شناخت و تجزیه و تحلیل



✓ ساختار اطلاعاتی مسئله

یک آرایه کمکی نیاز می باشد تا اطلاعات مرتب شده باشد.

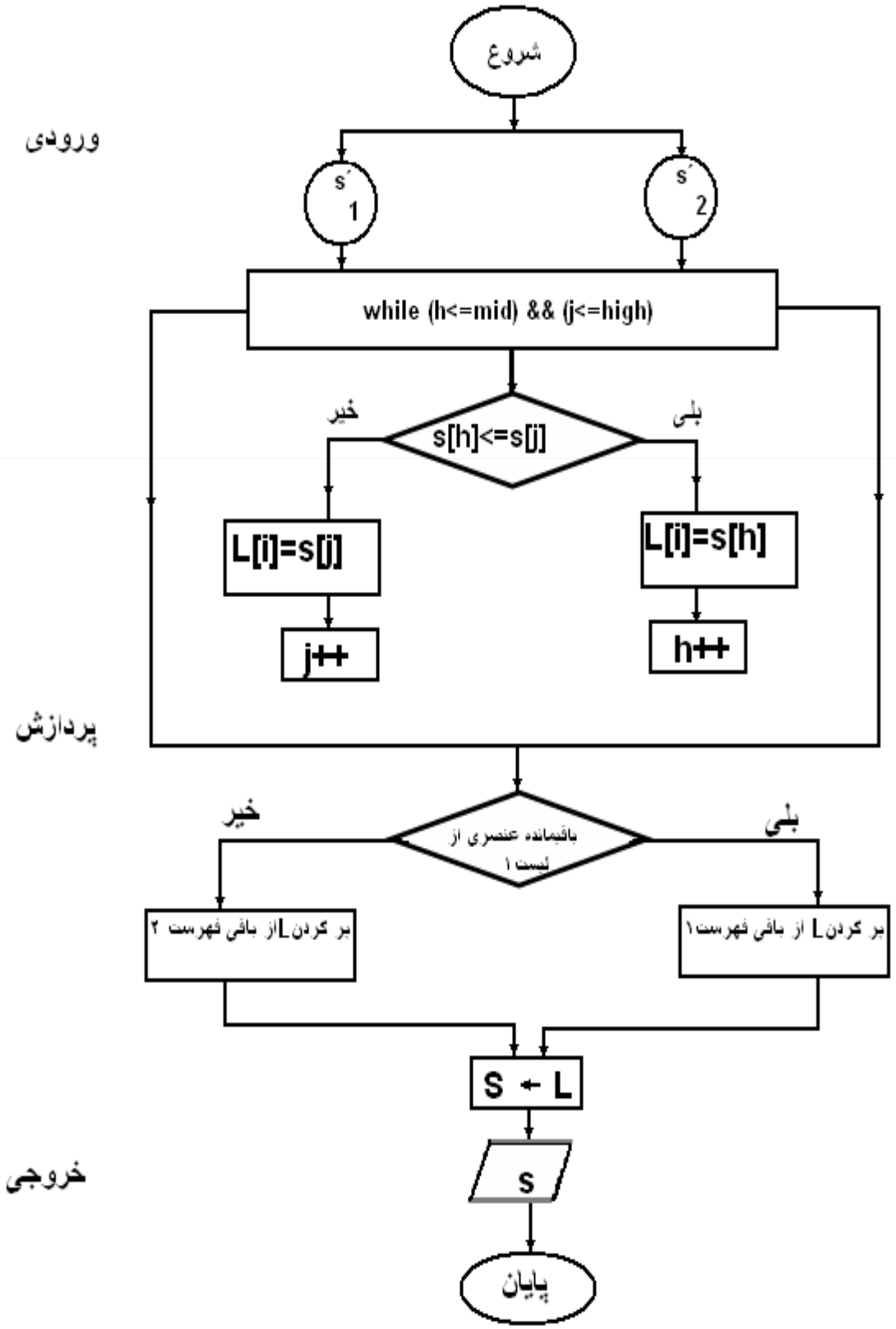


الگوریتم و فلوچارت مسئله:

- اگر مقدار لیست اول در اندیس h کوچکتر از نظیر آن در لیست دوم در اندیس j باشد اندیس h یکی جلومی رود و آرایه ادغامی Lازفهرست اول پرمی شود.در غیر این صورت آرایهL از فهرست دوم پر می شود.
- اگر عنصری از لیست اول نمانده باشد ،آرایه ی کمکی L ازعناصر باقی مانده ی فهرست دوم پر می شود.
- در غیر این صورت عناصر از لیست باقی مانده ی ۱ را در آرایه ی کمکی قرار می دهیم.
- اطلاعات آرایه کمکی L رادر آرایه ی اصلی ی معین کپی می نمائیم.

فلوچارت در صفحه بعد :

* کار حلقه پر کردن لیست L از کوچکترین عناصر در S_1' و S_2' است تا وقتی که یکی کاملاً پوشش داده شود.


شروع
ورودی
s´1
s´2
while (h<=mid) && (j<=high)
خیر
بلی
s[h]<=s[j]
L[i]=s[j]
L[i]=s[h]
j++
h++
پردازش
خیر
بلی
باقیمانده عنصری از لیست ۱
بر کردن L از باقی فهرست ۲
بر کردن L از باقی فهرست ۱
S ← L
خروجی
s
پایان

پیچیدگی الگوریتم مرتب سازی ادغامی :

$$T(n) = \begin{cases} \theta(1) & if\,(n=1) \\ 2T(n/2)+\theta(n) & if\,(n>1) \end{cases}$$

زمان ترکیب دو زیر رشته مرتب

زمان ۲ بار فراخوانی

حل:با یکی از روش های حل مانند روش تکرار وجایگذاری

$T(n) = 2T(n/2) + cn$
$= 2(T(n/2^2) + (n/2)) + cn$
$= 2(T(n/2^3) + cn/2^2 + cn/2) + cn$
$= \ldots$
$= 2(T(n/2^k) + cn/2^{k-1} + cn/2^{k-2} + \ldots) + cn$
$= \ldots$
$T(n) \in \theta(n \log n)$

نکته ی مهم:

Merge sort یک الگوریتم out place است یعنی در حافظه ی موجود عمل نمی کند و به حافظه ی دیگری نیاز دارد.

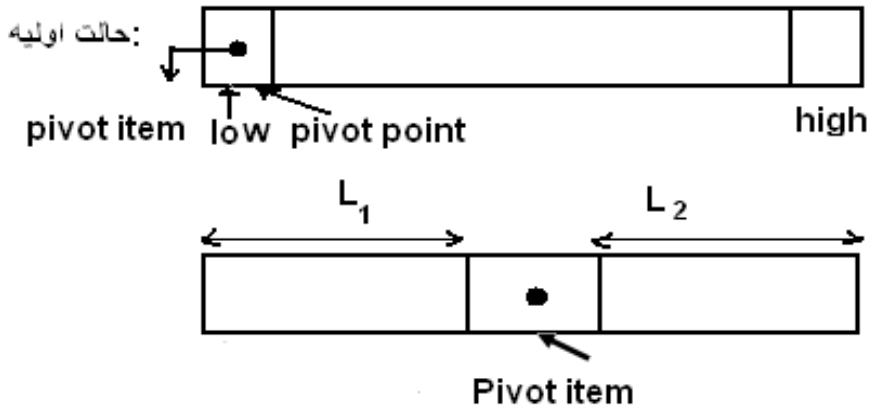
مرتب سازی سریع(quick sort)

- ✓ انتخاب یک عنصر به عنوان محور(pivot)معمولا" عنصر اول برای تقسیم به دو لیست در نظر گرفته می شود.
- ✓ قرار دادن عناصر کوچکتر از pivot در لیست سمت چپ و بزرگتر ازآن در لیست سمت راست و در حقیقت به معنی قرار گرفتن عنصرpivot در جای واقعی خودش.
- ✓ انجام کارهای ۱و۲ برای زیر فهرست های چپ و راست.

*از ویژگی های این روش :

- عدم الزام به برابری دو لیست.

- عدم نیاز به ادغام لیست های مرتب شده.



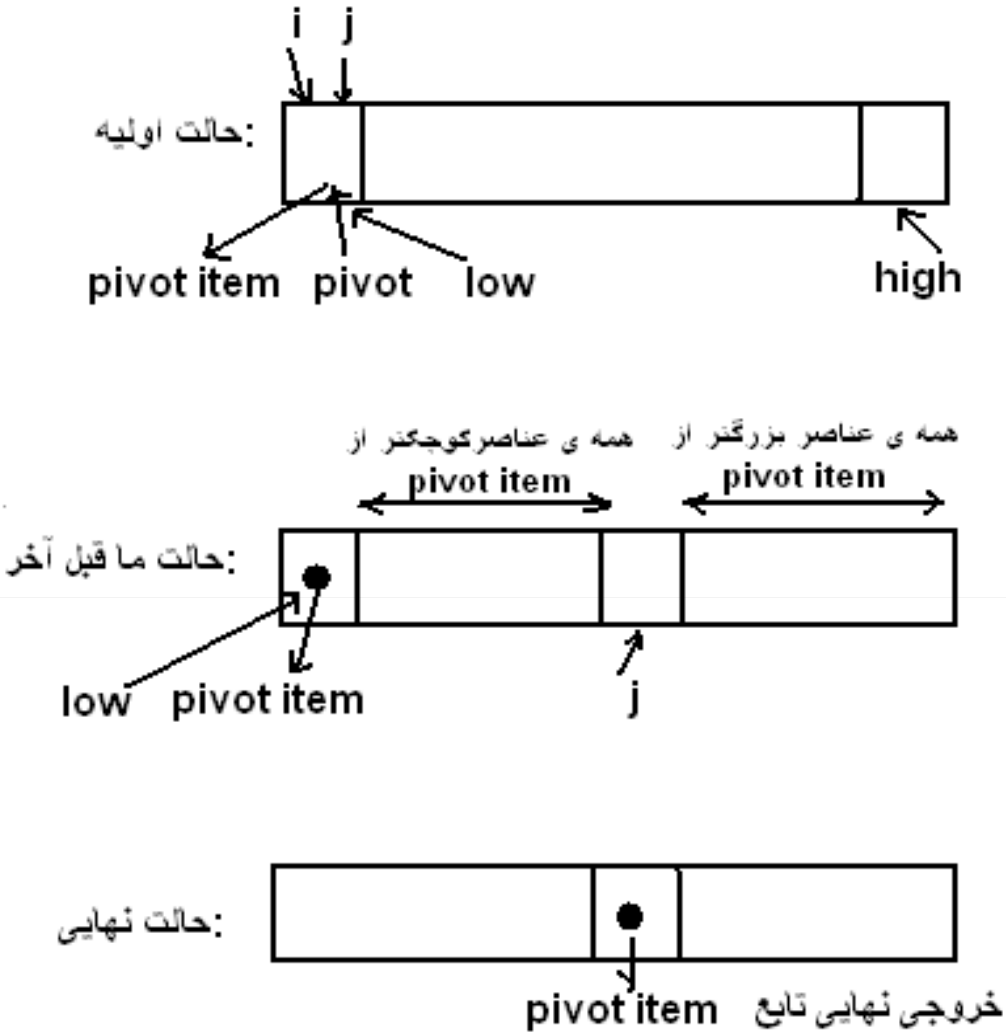
❋ لیست L الزاما به دوقسمت مساوی تقسیم می شود.

- در این روش سعی می کنیم عنصری از لیست را انتخاب کنیم و آن را در جای درست خود در لیست قرار دهیم در نهایت لیست به دو قسمت که الزاما" مساوی نیستند تقسیم می شود.همین کار را برای دو لیست چپ و راست انجام می دهیم تا در نهایت همه ی عناصر در جای خود مرتب قرار گیرند.

تمرین:این الگوریتم ها را به صورت فلوچارت در آورید(از طریق محصول Microsoft نرم افزار visio و فایل vsd.)

توضیح الگوریتم تقسیم بندی لیست:

- اندیس j به عنوان تعیین کننده ی نهایی عنصر pivot در لیست در نظر گرفته می شود.اندیس i از حالت اولیه شروع به حرکت می کند وهر جا عنصر مورد اشاره کوچکتر از عنصرpivot بود اندیس j یکی به جلو حرکت می کند ودر همان زمان محتویات اندیس i , j تعویض می گردد.در نهایت در مکان اندیس j ، تمام عناصر سمت راست به جز عنصر اول که خود pivot است کوچکتر ازpivot هستند همه ی عناصر با تعویض عنصر مکان j و عنصر low نتیجه حاصل می شود و pivot point به عنوان خروجی تابع تقسیم بندی پاس می شود.



*روش quick sort برای لیست مرتب بدترین حالت است.

*بهتر است عنصر pivot را به صورت random(تصادفی) انتخاب کنیم.

*می توانیم عنصر pivot را همان عنصر اول،وسط،آخر،1/4و...انتخاب کنیم اما فرض بهتر را بر این می گذاریم که به صورت تصادفی انتخاب شود. دقت کنید که با هر بار تغییر تمامی مراحل اندکی تغییر می یابد.

پیچیدگی الگوریتم quick sort:

- بدترین حالت:در هر بار تقسیم به دو لیست یکی تهی و دیگری شامل همه ی داده ها به جزعنصر pivot هستند.(به عبارتی داده ها مرتب هستند.)

بدترین حالت: $T(n)=T(0)+T(n-1)+(n-1)$

$$\Rightarrow T(n) = \begin{cases} T(0) & if\,(n=1) \\ T(n-1)+n-1 & if\,(n>=1) \end{cases} \Rightarrow T(n) \in \theta(n^4)$$

پیچیدگی الگوریتم quick sort در حالت متوسط:

در حالت متوسط pivot point همیشه و درهر تقسیم بندی توسط تابع partition در اول لیست قرار می گیرد. در بهترین حالت pivot point باید وسط لیست قرار بگیرد ولی احتمال اینکه در هر نقطه ای قرار بگیرد وجود دارد (با احتمال 1/n.)

$$T(n) = 1/n \sum_{p=1}^{n} T(p-1) + T(n-p) + (n-1)$$

تعداد عناصر در سمت چپpivot item

تعداد عناصر در سمت راستpivot item

زمان بخش بندی لیست Partition

زمان مرتب کردن سمت چپpivot point

P_ مقداری است که توسط تابع بر گردانده می شود.

اثبات:

1:nT(n)=(T(0)+T(1)+…+T(n-1))+(T(n-1)+T(n-2)+…+T(0))+n(n-1)

nT(n)=2(T(0)+T(1)+…+T(n-2)+T(n-1)+n(n-1) n→n-1;

2:(n-1)T(n-1)=2(T(0)+T(1)+…+T(n-2)+(n-1)(n-2)) 1-2;

nT (n)-(n-1) T (n-1) =n (n-1)-(n-1) (n-2) +2T (n-1)

$\Rightarrow nT(n)-(n+1)T(n-1) = n^2-n-n^2+3n-2=2(n-1)$

$$\Rightarrow \frac{T(n)}{n+1} - \frac{T(n-1)}{n-1} = \frac{2(n-1)}{n(n+1)}$$

$$\Rightarrow T_1(n) = \begin{cases} T_1(n-1) + \dfrac{p(n-1)}{p(n+1)} & if\,(n>=1) \\ 0 & if\,(n<1) \end{cases}$$

حل رابطه بازگشتی (با استفاده از روش کسر های جزئی):

$$T_1(n)-T_1(n-1)=\frac{2(n-1)}{n(n+1)}=\frac{A_1}{n}+\frac{A_2}{n+1}=\frac{A_1(n+1)+nA_2}{n(n+1)}$$

$$\Rightarrow (A_1+A_2)\,n+A_1=2n-2 \Rightarrow \begin{cases} A_1+A_2=2 \\ A_1=-2 \end{cases} \Rightarrow A_2=4$$

$$T_1(n)-T_1(n-1)=\frac{-2}{n}+\frac{4}{n+1}$$

$$T_1(n-1)-T_1(n-2)=\frac{-2}{n-1}+\frac{4}{n}$$

$$T_1(2)-T_1(1)=\frac{-2}{2}+\frac{4}{2+1}$$

$$T_1(1)-T_1(0)=\frac{-2}{1}+\frac{4}{1+1}$$

$$\Rightarrow T_1(n)-T_1(0)=-2\left(\frac{1}{n}+\frac{1}{n-1}+\ldots+\frac{1}{1}\right)+4\left(\frac{1}{n+1}+\frac{1}{n}+\ldots+\frac{1}{2}\right)=$$

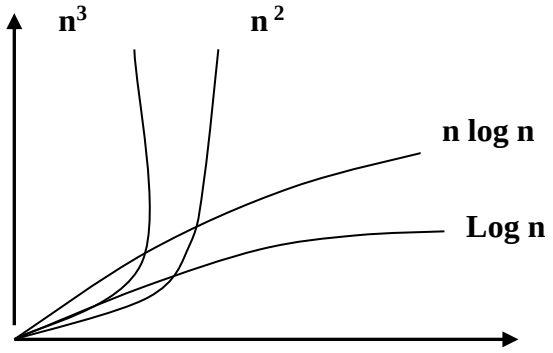
$$-2\left(\frac{1}{n}+\frac{1}{n-1}+\ldots+\frac{1}{1}\right)+\frac{4}{n+1}+4\underbrace{\left(\frac{1}{n}+\ldots+\frac{1}{2}+1\right)}_{s}=$$

$$-2s+\frac{4}{n+1}-4+4s=2s+\frac{4}{n+1}-4<2\ln(n)$$

$$S\sim\int_1^x \frac{1}{x}\,dx=\ln x\Big|_1^n=\ln x$$

$$\Rightarrow \frac{T(n)}{n+1}\in\theta\ln(n)\Rightarrow T(0)<=\theta(n\log n)$$

تمرین: برای n های ۱ تا ۱۰۰ توابع پیچیدگی زمانی زیر را در اکسل رسم کنید.



n	n^2	n^3	log n	n log n
1				
.				
.				
.				
100				

ضرب ماتریس ها به روش استراسن:

- ضرب دو ماتریس n*n

- در حالت عادی پیچیدگی زمانی $\theta(n^3)$

- در روش استراسن در حالت ماتریس 2*2:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}\begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}=\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

- برای n>2

تقسیم ماتریس به چهار زیر ماتریس.

استفاده از استراسن برای هر زیر بخش.

استفاده از رابطه ی4-5 برای محاسبه ی نتیجه ی ضرب دو ماتریس

به صورت بازگشتی.

تعداد عملیات محاسبه: ۷ضرب و۱۸ جمع

تحلیل پیچیدگی زمانی استراسن:

$$T(n)=\begin{cases} 7T(\frac{n}{2})+18\frac{n}{2} & n>1 \end{cases}$$

روش اصلی برای حل T(n):

$$a=7 \quad b=2 \quad F(n)=\frac{9n^2}{2} \quad G(n)=n^{\log_b a}$$

$\Rightarrow G(n)=n^{\log_2 7}\sim n^{2.81}>F(n)$

$\Rightarrow T(n)\in\theta\, G(n) \Rightarrow T(n)\in\theta(n^{2.81})$

نتیجه:کارایی بهتر از $o(n^3)$ در حالت ضرب عادی می باشد.

فصل سوم

روش حریصانه(Greedy)

الگوریتم حریصانه:

فلسفه ی این روش این است که برای حل مسئله در هر قدم یک گام به جواب نزدیک تر شویم و این گام بزرگترین گام در آن لحظه باشد به همین خاطر این روش صریح و ساده است.

دیدگاه حریصانه از دید فلسفی:

- دریافت بیشترین بهره در هر لحظه از زمان.
- زندگی در زمان حال.
- غنیمت شمردن رم.
- تمرکز بر زمان حال و شناخت مناسب از موقعیت حال وکسب بیشترین بهره از آن.
- عدم توجه به گذشته و آینده ی مسئله.

به دست آوردن درخت پوشای مینیمال (کمینه)minimum spanning tree:

ورودی : یک گراف .

خروجی : یک درخت از گراف به نحوی که مجموعه وزن یال های مینیمم باشد.

روش های بدست آوردن درخت پوشای مینیمال :

۱)الگوریتم پریم

۲)الگوریتم کروسکا ل

الگوریتم پریم :

۱) انتخاب و شروع از یک گره دلخواه.

۲) بررسی رئوس مجاور به مجموعه رئوس انتخاب شده و افزودن راسی که با یکی از رئوس موجود کمترین وزن یال را بسازد و در مجموعه حلقه ای ایجاد نشود.

۳) تکرار کار ۲ تا وقتی که رئوس گراف پوشش داده شود.

F=Ø;//مجموعه ی اولیه ی یال ها تهی است.

Y={v1}//از مجموعه ی گره ها اولین گره انتخاب می شود.

While (مسئله حل نشده است)

{ ۱:مرحله انتخاب

۲:بررسی امکان پذیری//انتخاب یک گره در گراف که به گره ی موجود نزدیکتر است.

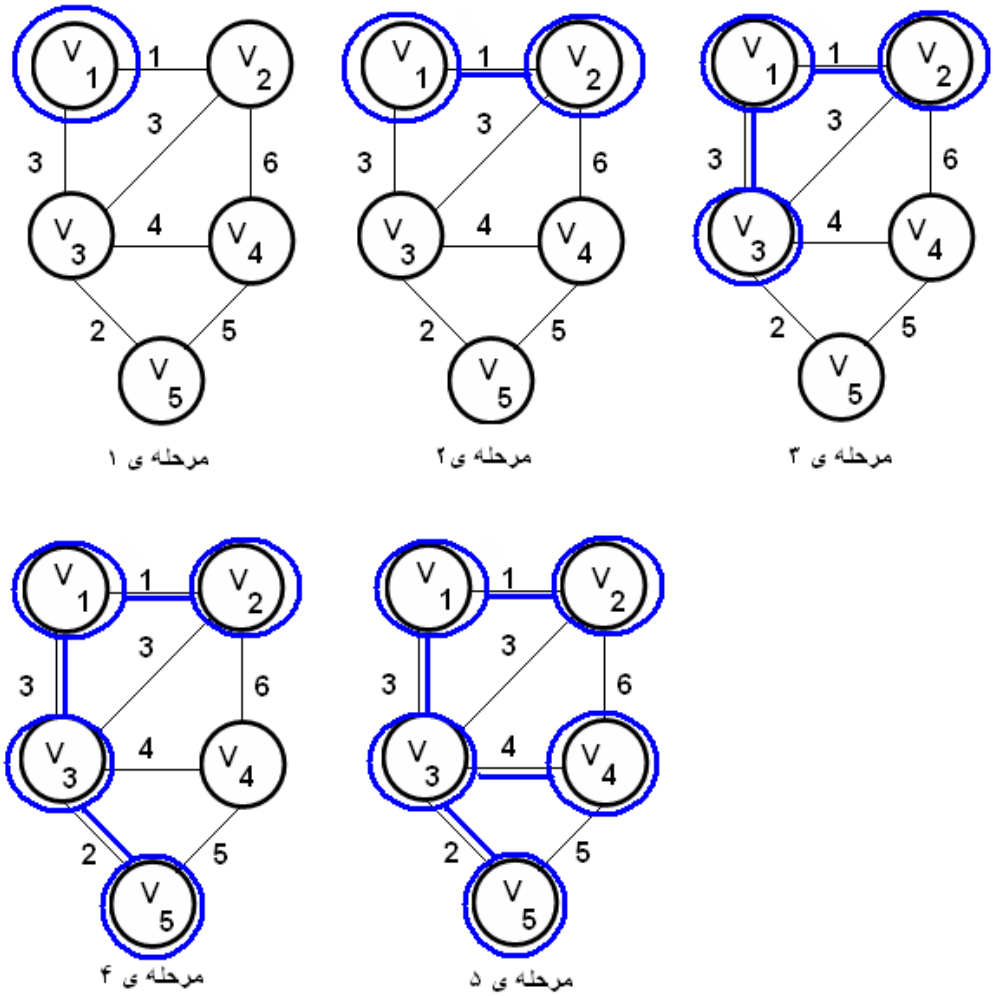
افزودن گره به مجموعه رئوس

افزودن یال به مجموعه یال ها

بررسی انتخاب // اگر همه ی رئوس پوشش داده شود. (Y=V)

مسئله حل است. }

مثال:



توجه:درخت پوشای مینیمال منحصر به فرد نیست.(در مرحله ۲ یال V2 V3 نیز می تواند انتخاب شود.)

توجه:ممکن است وزن مجموع یال ها نیز متفاوت باشد:Local minimum

*در الگوریتم ژنتیک سعی بر این است که با ضربه یا (**impulse(local minimum**

مواجه شویم.

الگوریتم کروسکال :

۱) مرتب کردن یال ها به ترتیب وزن به صورت صعودی.

۲) انتخاب یال با کمترین وزن تا اندازه ای که دور ایجاد نشود.

۳) تکرار کار ۲ تا وقتی که گره ای باقی نماند.

مراحل الگوریتم کروسکال:

۱) ایجاد زیر مجموعه های مجزا از هم از v با یک راس(initial).

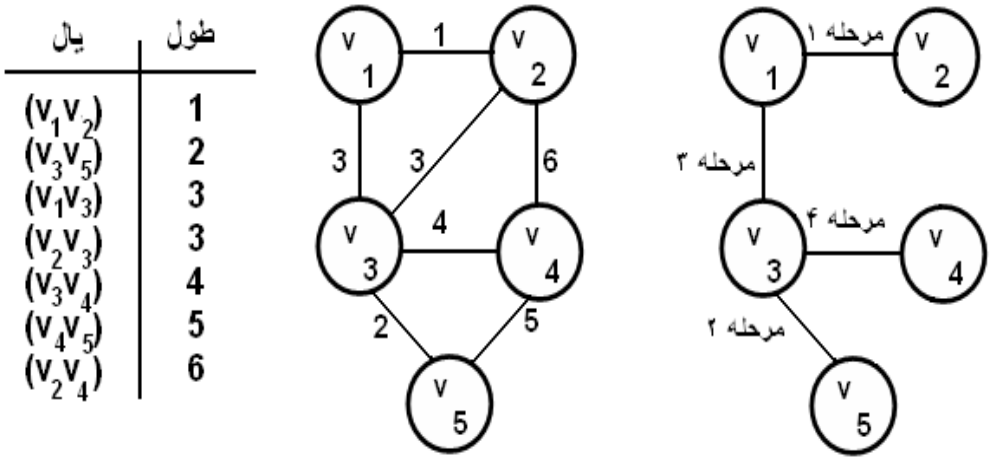
۲) مرتب کردن یال ها به صورت غیر نزولی بر حسب وزن یال ها(heap).

۳) یافتن یال برای اضافه کردن به مجموعه جواب به شرط این که یال دو زیر مجموعه جدا از هم را به هم وصل کند.(find)

۴) ادغام دو زیر مجموعه (merge).

۵) تکرار مراحل بالا تا ادغام تمام زیر مجموعه ها.

مثال:



یال	طول
(V_1V_2)	1
(V_3V_5)	2
(V_1V_3)	3
(V_2V_3)	3
(V_3V_4)	4
(V_4V_5)	5
(V_2V_4)	6

پیچیدگی زمانی الگوریتم های پریم و کروسکال:

الگوریتم پریم: $T(n) \in \theta(n^2)$

الگوریتم کروسکال: $T(m,n) \in \theta(n^2 \log n)$

در یک گراف متصل در حالتی که درخت متراکم یا heap است $n-1 \le m \le \frac{n(n-1)}{2}$

نکات تستی:

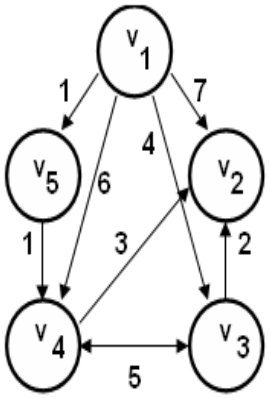
الگوریتم	پریم	کروسکال
گراف کامل	$\theta(n^2)$	$\theta(n^2 \log n)$
heap(گراف متراکم)	$\theta(n^2)$	$\theta(n \log n)$

*مقداری که از روش کروسکال به دست می آید منحصر به فرد است و مینیمم مطلق است.

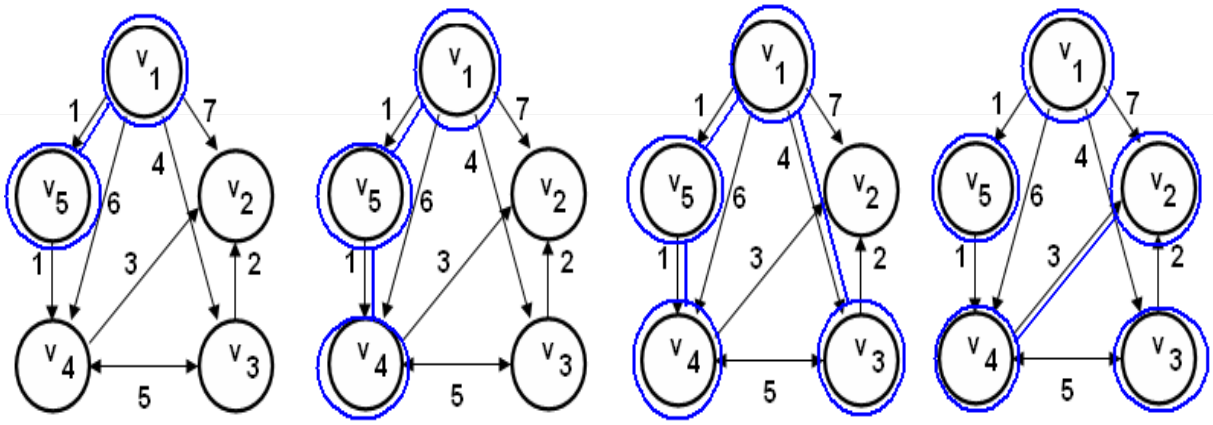
الگوریتم دیکسترا:

یافتن کوتاهترین مسیر ها از مبدا واحد به مقصد های متفاوت در یک گراف جهت دار و موزون (کاربرد گسترده در شبکه های کامپیوتری و کاربردهای نوین در آموزش مجازی).

مثال:تمام مسیر ها از V_1 به بقیه ی رئوس را بیابید.



ورودی



مرحله ی ۱ مرحله ی ۲ مرحله ی ۳ مرحله ی ۴

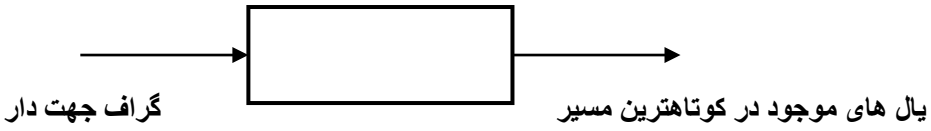
توجه:در این الگوریتم مسیر یال ها بسیار اهمیت دارد ودر خلاف جهت یال ها نمی توان حرکت کرد.

* در این روش مقداری که در هر مرحله انتخاب می کنیم باید نسبت به مبدا مقایسه گردد.علاوه بر آن چون جهت دار است باید به جهت آن نیز دقت شود.

مسئله:تعیین کوتاهترین مسیر از v_1 به همه رئوس دیگر درگراف موزون جهتدار

ورودی:عدد صحیح $n>=r$ و یک گراف جهت دار،موزون و متصل حاوی n راس (این گراف توسط آرایه ی دوبعدی w نشان داده می شود که سطر و ستون آن از1تا n اندیس گذاری شده اند و در آن w[i][j] وزن ما بین راس i ام به راس j ام است.

خروجی:مجموعه یال های F حاوی یال های موجود در کوتا هترین مسیر.



مراحل:

۱) مقداردهی اولیه برای تمام گره ها :مقداردهی طول آن مسیر به وزن یال ها از مبدا v_1.

۲) اضافه کردن n-1 یال به y

- بررسی این که هر یال دارای کوتاهترین مسیر است.

- بررسی هر یال که در y نیست کوتاهترین مسیر بروز شود.

*این الگوریتم فقط یال های کوتاهترین مسیر را تعیین می کند و طول یال را به دست نمی دهد.این طول را می توان از یال ها به دست آورد.

پیچیدگی زمانی الگوریتم دیکسترا:

$T(n) \in \theta(n^2)$

مسئله ی خرد کردن پول:

تعداد سکه	ارزش سکه
۱	نیم دلار
۱	۲۵ سنت
۲	۱۰سنت
۱	۵ سنت
۱۰	۱سنت

ورودی:تعدادی سکه (جدول روبرو)

خروجی:پس دادن بقیه پول ۴۷ سنت مشتری با حداقل سکه

جواب بهینه ی اصلی:

- انتخاب های حریصانه بهینه ی محلی.

- هر زیر مسئله دارای جواب بهینه است.

Objective (هدف)	Solution (حل)	Feasible (امکان پذیری)	Select (انتخاب)	میزان پول باقی (سنت)	سکه انتخابی (سنت)	شماره مرحله
-	-	-	+	47-50<0	50	1
-	-	+	+	47	~~50~~	2
-	-	+	+	47-25=22	25	3
-	-	+	+	22-10=2	25,10	4
-	-	+	+	12-10=2	25,10,10	5
-	-	-	+	2-5<0	25,10,10,5	6
-	-	+	-	2	25,10,10,~~5~~	7
-	-	+	+	2-1=1	25,10,10,1	8
+	+	+	+	1-1=0	25,10,10,1,1	9

مجموعه ورودی

```
Set   greedy – applying (c)
{
    C= {1,2,5,40,20,25,50 }  ;
    S=[0] ;
While ( ! solution (s)  &&  c!= ∅) //
```

تا وقتی که پول کاملا خرده شده و مجموعه هم خالی نیست

```
 {
 X=select (c) ; //
```

انتخاب یکی از اعضای مجموعه ورودی

```
 C=c-{x} ; //
```

کم کردن آن عنصر از زیر مجموعه ورودی

```
If (faceibele (s,x))
    S=s+{x} ; //
```

* افه کردن عنصر به مجموعه جواب در صورت شدنی بودن(faceibllity)

*

} * یک عنصر به مجموعه عنصر[x] اضافه می شود

```
If solution (s) ;
    Return s ;
Else
        Return 0 ;
}
```

مسئله کوله پشتی :

مثال :

Pi ارزش شیء i

Wi وزن شی i

ورودی :p , w(دویست شامل ارزش و وزن گیاه)

خروجی : فهرستی از اشیاء انتخاب شده با بیشترین ارزش که از ظرفیت کوله پشتی تجاوز نکند .

Xi	pi	wi	pi/wi
X1	**8**	**16**	**8/16**
X2	**5**	**15**	**5/15**
X3	**15**	**25**	**15/25**
X4	**10**	**8**	**10/8**
X5	**20**	**15**	**20/15**

Max تابع هدف : $\sum_{I=1}^{N} pi \; xi$

ارزش عنصر i ام(abjective)

میزان انتخاب شده از عنصر I ام ،xi می تواند صفر ، یک یا کسری بین wi <1 باشد

وزن عنصر I ام

(constraints)محدودیت ها : $\sum_{I=1}^{N} Wi \; xi < w$

ظرفیت کوله پشتی

3 نگرش برای حل حریصانه :

جواب ها الزاما بهینه نیستند

۱. انتخاب یک شی (اشیاء) با بیشترین سود جهت سود ماکزیمم
۲.انتخاب اشیاء : کمترین جهت برآوردن محدودیت ظرفیت کوله پشتی
۳. انتخاب اشیاء با در نظر گرفتن نسبت ارزش/وزن(pi/wi)

جواب بهینه می دهد

اگر جدول مسئله را برحسب pi/wi و نزولی مرتب کنیم :

معیار انتخاب selection feceible soluction

Xi	pi	wi	pi/wi
X5	20	15	20/15
X4	10	8	10/8
X3	15	25	15/25
X1	8	16	8/16
X2	5	15	5/15

Xi	X1	X2	X3	X4	X5	w
	0	0	0	0	1	40-15=25
	0	0	0	1	1	25-8=17
				1	1	17-25 <0
			17/25	1	1	12-17/25*25

$$\sum_{I=1}^{S} pi\ xi = 0*8+0*15+17/25*15+1*10/10+1*20/20=40.1$$

چون 17<25 است لذا تنها قسمتی از آن انتخاب می شود .

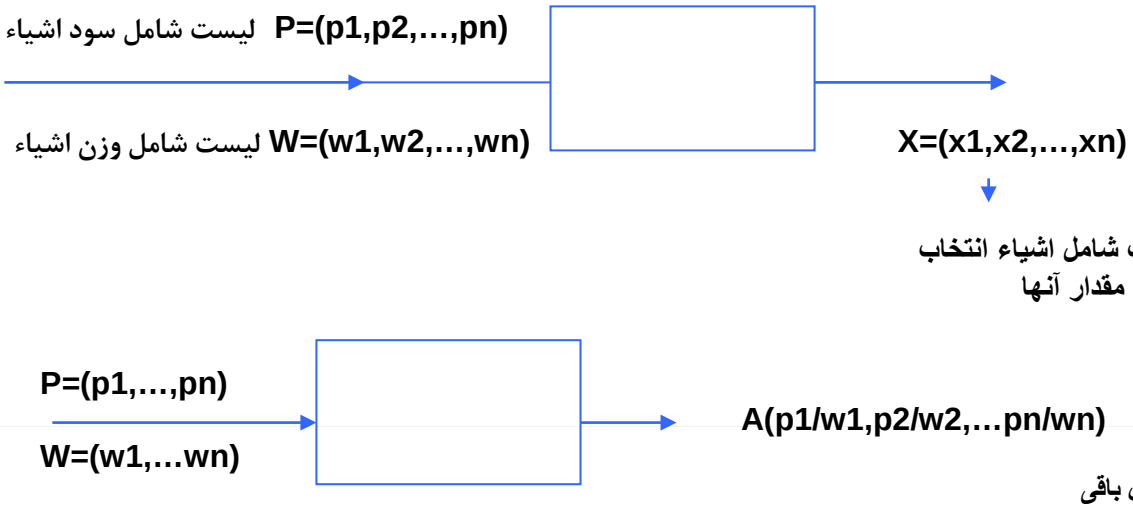
ظرفیت کوله پشتی

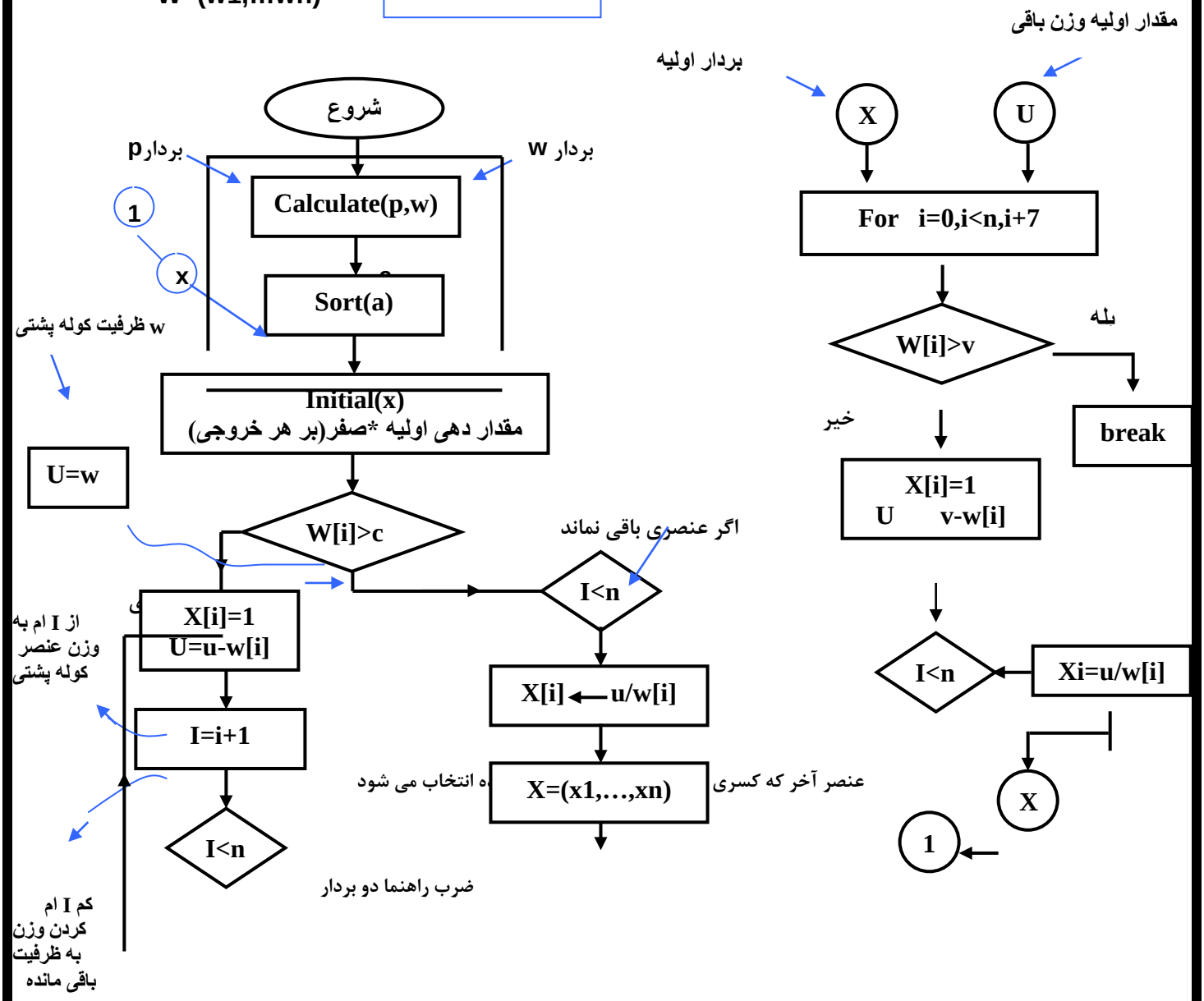
❊ شبه کدc در صفحه ۱۶۵❊

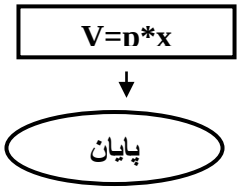
تحلیل پیچیدگی الگوریتم کوله پشتی :

$T(n) \in \theta(n\log n)$

الگوریتم کوله پشتی (با استفاده از روش حریصانه :







مسئله زمانبندی :

دو گونه زمان بندی داریم :

۱.کمینه سازی کل زمان برای انتظار کشیدن وسرویس دهی کارها (زمان بودن در سیستم)
(مثال :آرایشگاه (اصلاح سر – اصلاح ریش- اصلاح سر کم پشت)

۲.زمان بندی با مهلت معین

T1=5 , t2=10 , t3=4 → * در تمام مراحل باید به این توجه کنیم

مثال : 3!=3*2*1=6

کار	زمان بودن در سیستم
1	5 (زمان سرویس دهی)
2	5 (زمان انتظار در کار ۱) +(زمان سرویس دهی)10
3	5 (زمان انتظار در کار ۱)+(زمان انتظار در کار ۲)+ + زمان سرویس دهی

5+(10+5)+(5+10+4)=39

زمان بندی ۶حالت	کل زمان بودن در سیستم
(1,2,3)	5+(5+10)+(5+10+4)=39
(1,3,2)	5+(5+4)+(5+4+10)=33
(2,1,3)	10+(10+5)+(10+5+4)=44
(2,3,1)	10+(10+4)+(10+4+5)=43

جواب بهینه*

(3,1,2)

(3,2,1)

T1=3 t2=2 t3=1

4+(4+5)+(4+5+10)=32

4+(4+10)+4+10+5)=37

بهترین حالت با کمترین زمان

* در واقع در هر مرحله ترتیب را تغییر می دهیم و برای مثال در آخری زمان انجام انجام کار3 همان t1 خواهد شد زمان انجام کار دوم 2 همان t2 خواهد شد وزمان انجام کار1 همان t3 خواهد شد .

نکات :

۱- زمان بندی (3,1,2) بهینه است .

۲- تابع زمانی در نظر گرفتن همه زمان بندی ها فاکتوریل است .

۳- استفاده از الگوریتم حریصانه برای انتخاب کارهای با زمان با زمان سرویس کوچکتر شبه کد در صفحه کتاب

تعمیم الگوریتم فوق برای چند سرویس دهنده :

سرویس دهنده1 به کارهای 1,(1+m),(1+2m),(1+3m)

سرویس دهنده 2 به کارهای 2,(2+m),(2+2m),(2+3m)

سرویس دهنده I به کارهای I,(i+m),(i+2m),(i+3m)

سرویس دهنده m به کارهای m,(m+m),(m+2m),(m+3m)

سرویس می دهد .

توجه : الگوریتم زمان بندی با مهلت گفته نشده صفحه 173,174

کد گزاری هافمن :

مسئله کلی : تولید رشته ای از کارکترها به رشته ای دیگر :

۱- روش asci: تبدیل کارکتر ها به یک تعداد بیت ثابت مثلا 8 بیتی در این

روش هر8 بیت یک کارکتر در نظر گرفته می شود.

A → 00100001
B → 00100010

۲- روش هافمن : اگر برخی کارکتر ها زیادتر باشند بهتر است روش در نظر بگیریم که آن کارکتر در متن کد شده با در نظر گرفتن 2 نکته :

۱. طول رمزها برای کاراکترها با تکرار بیشتر کمتر باشد .
۲. کدهایی انتخاب شوند که به عنوان بخش ابتدایی که کاراکتر دیگری نباشند.

مث

A	B	C	D	E	F
1600	600	100	100	1600	1700

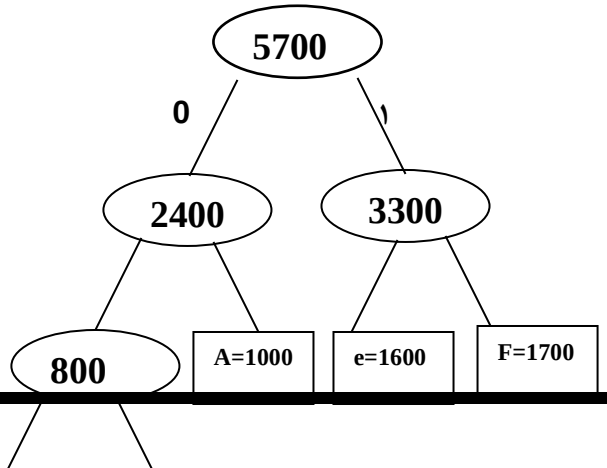
شروع :

۱. مرتب کردن کارکتر ها با درخت ها به صورت تکرار صعودی
۲. ادغام دو کارکتر اول و تشکیل یک درخت با تکرار مجموع دو کاراکتر
۳. باقی ماندن یک مجموعه جدا

روش انجام کار :

C=100	D=100	B=600	A=1600	E=1600	F=1700

توجه :❋ در روش حریصانه از houk استفاده می شود .



C=0000
D=0001

1 ۰

0 1

1 ۰

کد نوشته شده در کتاب اشتباه

است منظور همان شکل

صفحه 175 ۱ ۰

کاراکتر	C	D	B	A	E	F
کد هافمن	0000	0001	001	01	10	11
تکرار	100	100	600	1600	1600	1700
تعداد بیت ها	4	4	3	2	2	2
تعداد کل بیت ها	400	400	1800	3200	3200	3400

فصل چهارم

برنامه نویسی پویا

برنامه سازی پویا

برنامه سازی پویا از ارایه استفاده می کند و پایین به بالاست (اما روش تصمیم حل برعکس از بالا به پایین است .)

از **STEAK** استفاده می کند.

ضریب دو جمله ای :

با استفاده از تقسیم وحل :

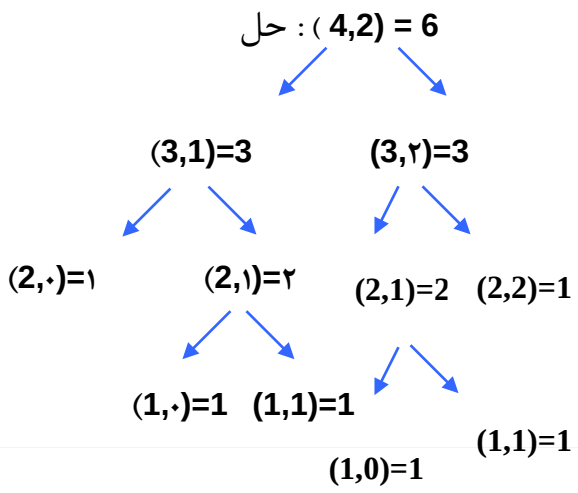
ورودی : اعداد صحیح و مثبت N,K که در آن K<N

خروجی :BIN ضریب دو جمله ای (N,K)

شبه کد در صفحه 203 کتاب

```
Int bin (int n , int k)
{
```

```
  If (k=0  or  n=k)      return 1 ;
  Else   return bin (n-1 , k-1) + bin (n-1 , k) ;
}
```



با استفاده از برنامه سازی پویا : صفحه 205کتاب

```
Int bin (int n , Int k)
{
  Int I , j ;
  Int B[n] [k] ;
  For (i=0 ; i<=n ; i++)
     For (j=0 ; j<=min(I,k) ; j++)
        If  (j==0  //  j==1)
          B[i] [j] = 1 ;
        else          B[i] [j] = B[i-1] [j-1] + B[i-1] [i] ;
   Return  B[n] [k] ; }
```

N K

N \ K	0	1	2	3	4
0	۱				
1	۱	۱			
2	۱	۲	۱		
3	۱	۳	۳	۱	
4	۱	۴	۶	۴	
5	۱	۵	۱۰		
6	۱	۶			

N نباید از K کوچکتر باشد

(2,1)

(1,0) (1,0)

چون در آرایه قبلا محاسبه شده

از همان ۲ استفاده می کنیم .

جواب ۲و۴ که در مسئله قبل از طریق تقسیم وحل به این جواب رسیده بودیم.

جواب تکی یک پشت'راه حل متفاوت است.در مرحله قبل از seaek استفاده کردیم.پس در روش از craray استفاده کردیم که بدین سبب مراحل که در از دست داد یم ونیازی به محاسبه ی مقادیر تکراری نیست .

(3,1)

(2,1) (2.۱)

روش تقسیم و حل :

۱- نوشتن یک تابع (رابطه ی بازگشتی)

۲- استفاده از تابع بازگشتی از بالا به پایین برای محاسبه

۳- ساختار اطلاعاتی : پشته

۴- علت ناکارآمد : محاسبات تکراری

روش برنامه سازی پویا :

۱- نوشتن یک رابطه ی بازگشتی

۲- محاسبه ی مقادیر خانه های آرایه از پایین به بالا (برای آرایه از f(b) ها)

$$T(n) = \begin{cases} (n-1 , k-1) + (n-1 , k) \\ k = n \quad =1 \\ \mathbf{K = 0 \quad =1} \quad \text{روش تقسیم و حل} \\ \text{روش برنامه نویسی پویا} \end{cases}$$

$$t(n)=\binom{n}{k} -1$$

$$t(n)=\theta (nk)$$

مسئله :جمله n ام دنباله ی فیبوناچی

ورودی ها :n>=1

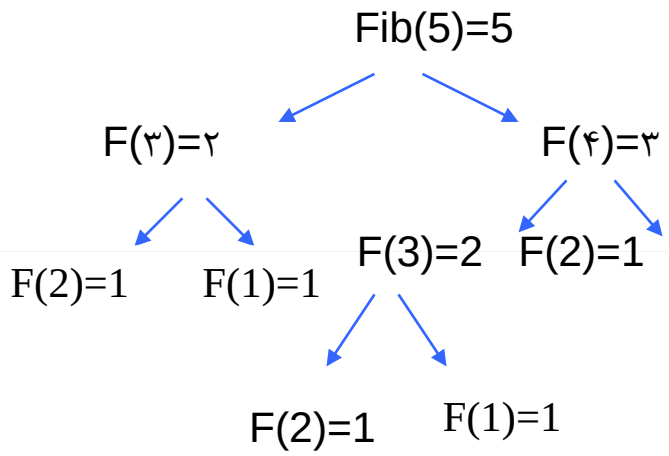
خروجی ها :جمله n ام دنباله ی فیبوناچی

F(0)=0 , f(1)=f(2)=1
F(n)= fib(n-1) + fib(n-2)

از روش تقسیم و حل : ساختار پشته

۱	۲	۳	۴	۵	۶	۷
۱	۱	۲	۳	۵	۸	۱۳

علت ناکارآمد : عملیات تکراری و محاسبات تکراری



$$T(n)=\begin{cases} T(n-1) + t(n-2) & n>=2 \\ C & n=1 \end{cases}$$

$T(n)= t(n-1) + t(n-2) > t(n-2) + t(n-2) \longrightarrow 2t(n-2) \rightarrow$ پیچیدگی الگوریتم فیبوناچی 2^n/2

روش برنامه سازی پویا :

F(n) = f(n-1) + f(n-2)

0	1	1	2	3	5	8	13	.....	
0	1	2	3	4	5	6	7		n

فصل پنجم

تکنیک عقب گرد

تکنیک عقبگرد back tracking

مسئله ی ۴ وزیر : ۴ وزیر را در یک صفحه شطرنج ۴*۴ بنحوی قرار دهید که هیچ ۲ وزیری یکدیگر را تهدید نکنند .

حل : روش متعارف و عادی

شرط تهدید نکردن ۲ وزیر:
۱. ۲ وزیر در یک سطر نشیند
۲. ۲ وزیر در یک ستون نشیند.
۳. ۲ وزیر در یک سطر نشیند.

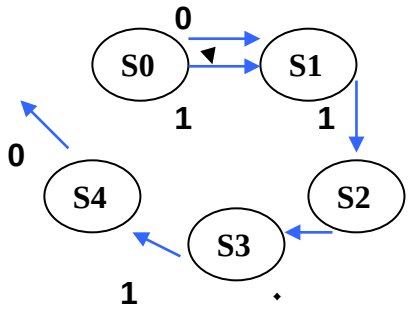
	1	2	3	4
1	0			
2	0			
3		0		
4		0		

تعداد کل حالات

16*15*14*13=16!/12!

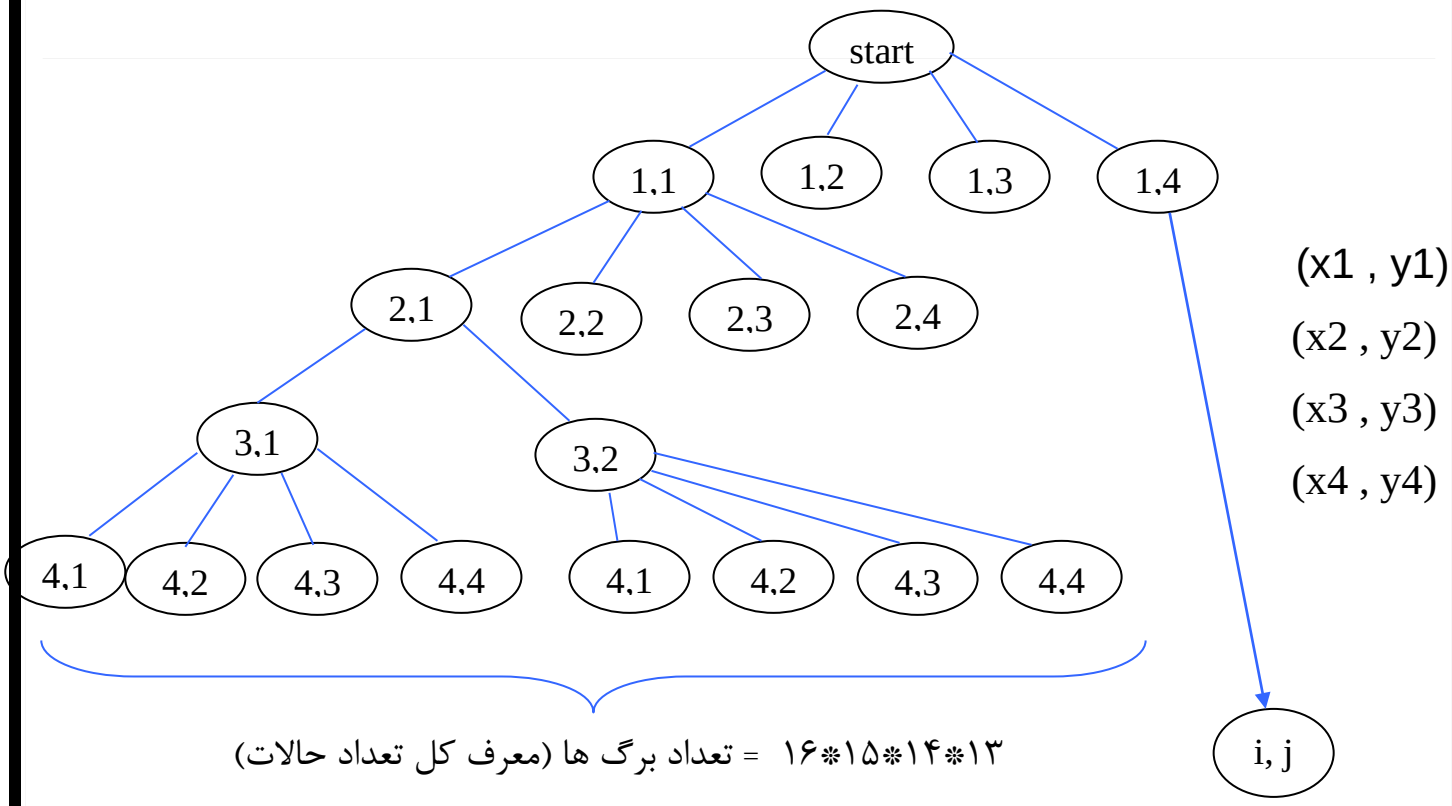
تعریف حالت (state):

اگر متغیرهای مسئله x1,.....,xi باشند رابطه (x1,.....,xn) معرف حالت است . حالت مسئله در حقیقت وضعیت فعلی آن سیستم را نشان می دهد ، از مفاهیم مهم در نظریه سیستمم ها می باشند با تغییر هر مقدار از متغیر ها حالت مسئله هم تغییر می کند .



درخت فضای حالت :

معرف وضعیت متغیرهای مسئله در تمام حالات می باشد ،با تغییر مقدار متغیرها ، حالت مسئله نیز تغییر می کند ، یک حالت خاص در حقیقت یک مسیر از راس درخت تا یکی از گره ها می باشد .



تعداد برگ ها (معرف کل تعداد حالات) = ۱۶*۱۵*۱۴*۱۳

به معنی آن است که وزیر در سطر i ام و ستون j قرار دارد .

بردار حالت : ((1,1) (2,1) (3.2) (4,2))

	1	2	3	4
1	0			
2	0			
3		0		
4		0		

تیپ مسائل تصمیم گیری ← ساختار اطلاعاتی درخت tree desigion

(Game theory) نظریه زبان ها ← ytratoyy

MCDM : multiple cviteria desigion malaing

تعداد حالات بسیار زیاد است ← احساس نیاز به تعریف جدیدی از متغیرهای حالت برای کم کردن حالت ها

پیش فرض : هیچ ۲ وزیر در یک سطر یا ستون نباشند ← عناصر متغیر حالت تک بعدی می شوند .

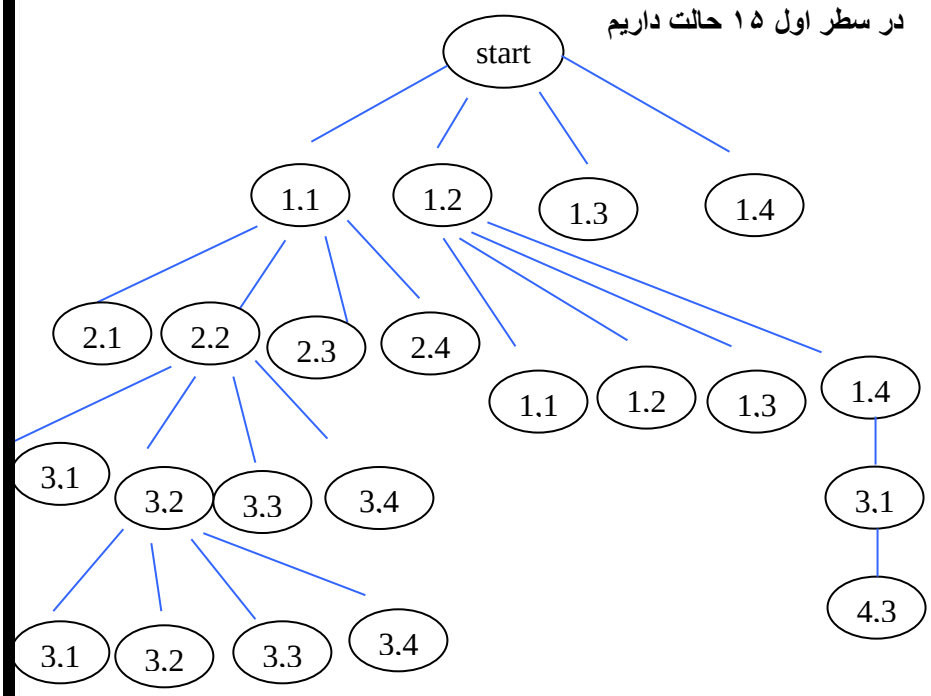
بردار حالت برای مثال بالا با این فرض (1,1,2,2)

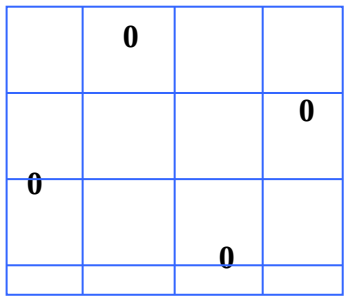
شماره ستون با این پیش فرض که شماره سطری یکی نیستند

درخت حالت با تعریف متغیرهای جدید :

(1,1) (1,2) (2,1) (2,3)

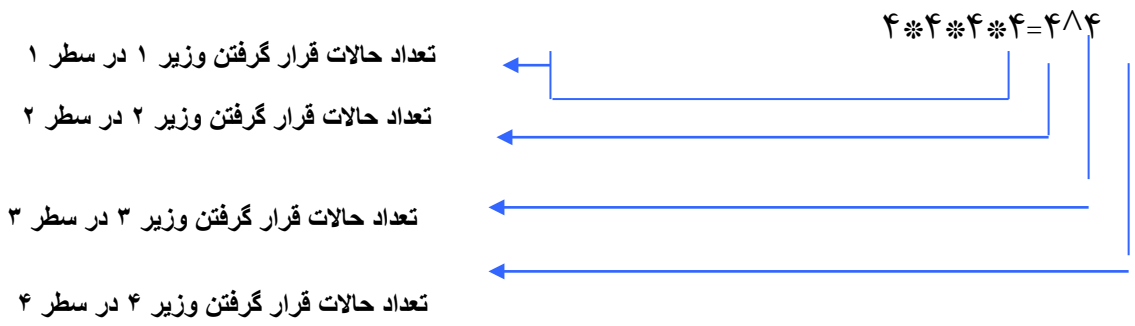
در سطر اول ۱۵ حالت داریم



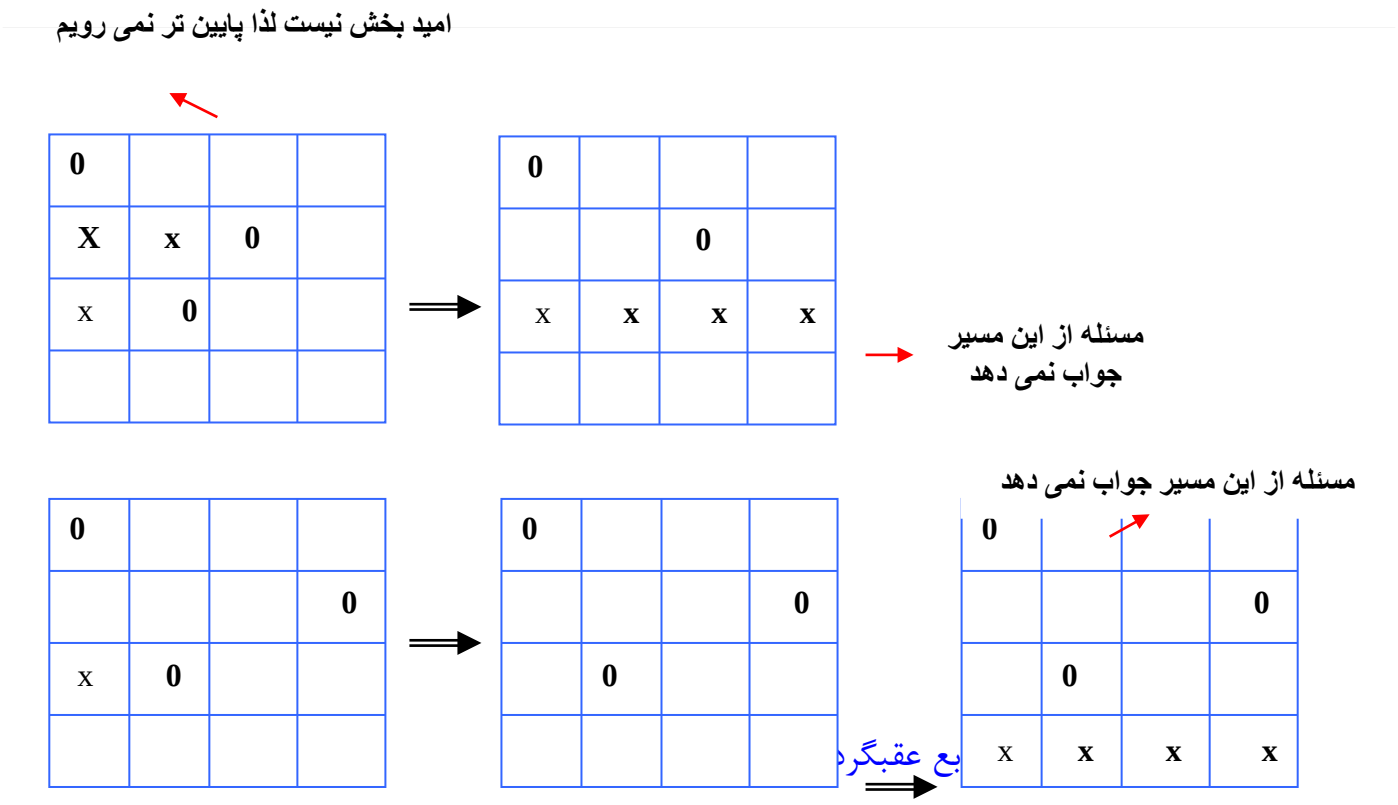


کتاب255صفحه

تعداد کل حالت ها (برگ ها)

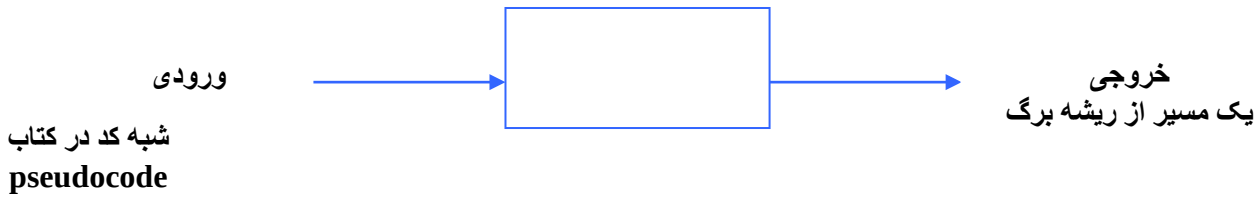


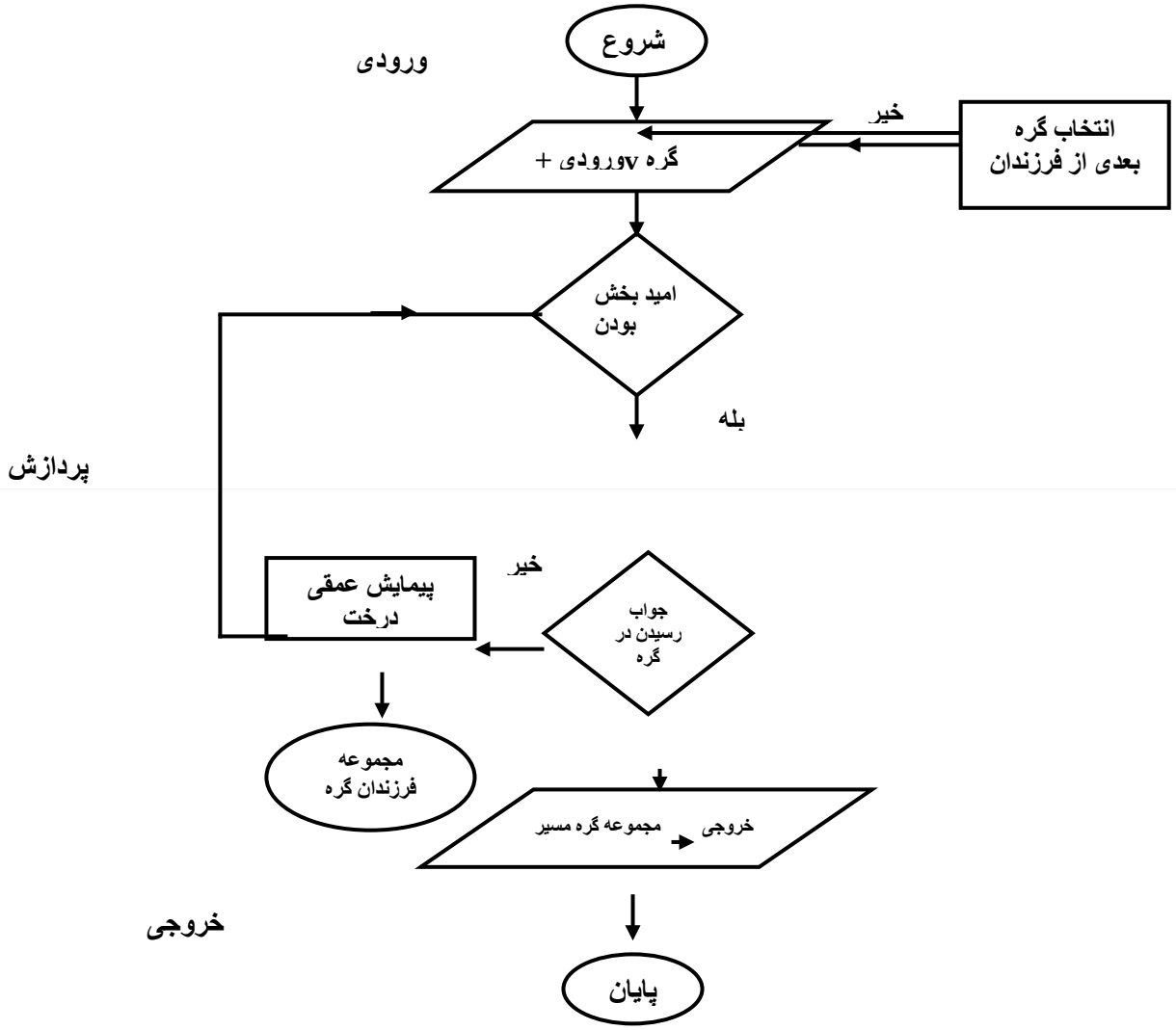
روش عقبگرد حالت خاصی از پیمایش عمقی است که هر گاه گره امید بخش نبود دیگر در آن مسیر به عمق بیشتری نمی رویم و به عبارتی پایین تر نمی رویم .



ورودی : درخت حالت

خروجی : یک مسیر از ریشه به برگ با گره های امید بخش





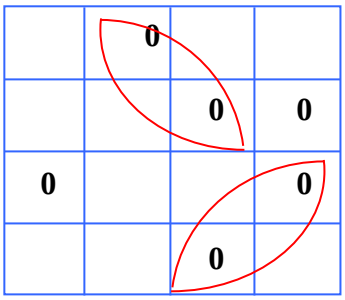

```
Void  baek track  (node v)
{
   Node u ;
   If  promising (v) ;
      If  (ther is a solution at v)
       Write the solution ;
       Else

For  (each child u of v)
     Baek track  (u) ;
                    }
```

بررسی امید بخش نبودن در مسئله ۴ وزیر (n) (x1,x2,….xn) یک جواب از یک جواب از فضای حالت (شماره سطر وزیر معین و شماره ستون وزیر معین)

1-بررسی ستونها	col(i)=col(k) یا Xi=Xk	j=l شماره ستون ۲وزیر یکی است
۲-بررسی سطرها	col(i)-col(k)=i-k* یا xi-xk=i-k Col(i)+col(k)=k+i* یا Xi+Xk=k+i	i-j=k-l i+j=k+l i-k=k-j **۲وزیر در سطر همدیگر را تهدید می کنند**

یک جواب از فضای حالت (x1,x2,x3,x4)=(2,4,1,3)

فرمول صفحه ۲۵۷ کمی مبهم است اصلاح شود .



صورت مسئله : تا این مرحله مقادیر x1 تا k-1 مشخص شده اند می خواهیم ببینیم که آیا مقدار k امیدبخش است یا نه

مقادیر قبلی

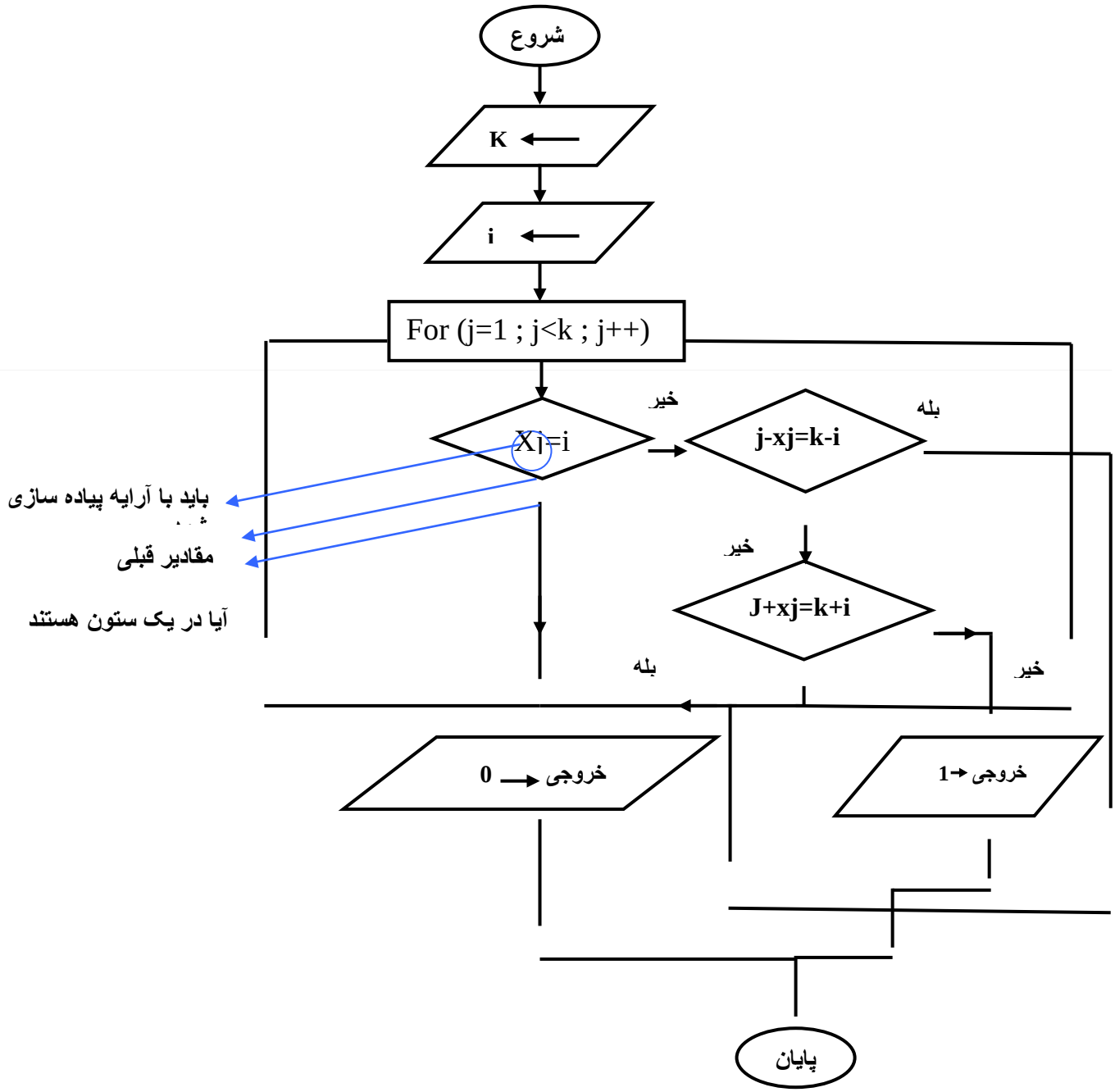
(x1,x2,……,xj,……,k,……)

j **شماره ستون فعلی**

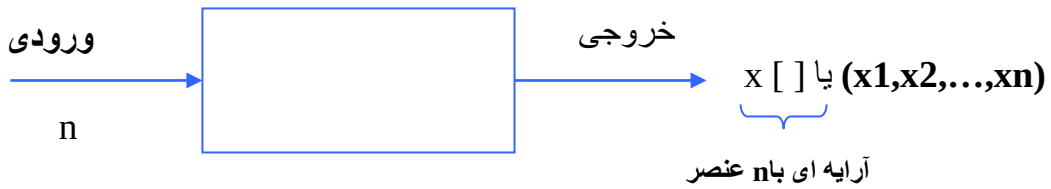
توجه شود در صفحه قبل { l شماره سطر اول j شماره ستون اول است. K شماره سطر دوم l شماره ستون دوم }

الگوریتم امید بخش بودن برای مسئله n وزیر: promising (k,i)

شماره سطر شماره ستون

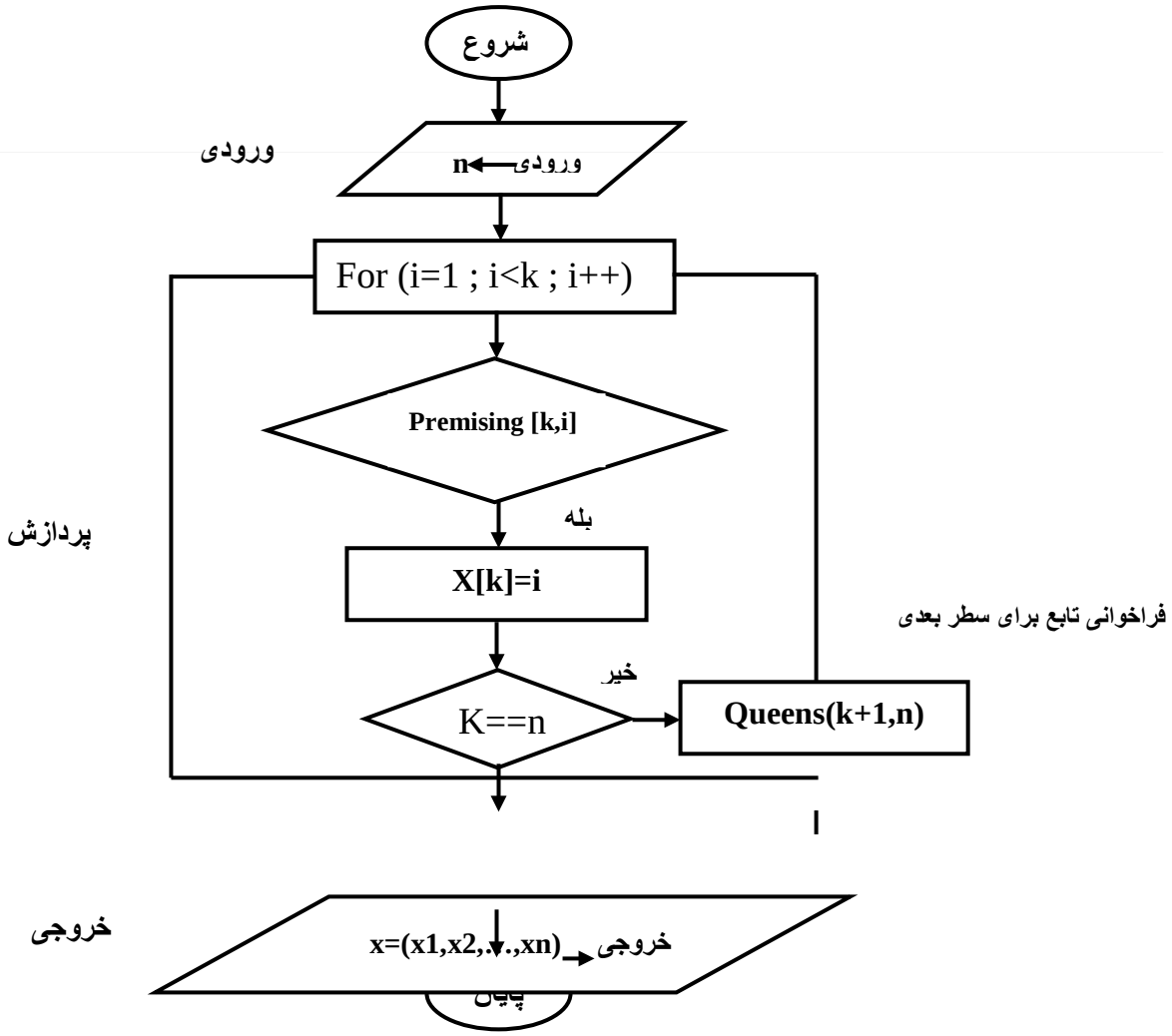


الگوریتم عقبگرد برای مسئله n وزیر:



Queens (k , n)

تعداد وزیرها شماره سطر وکد شروع کار یک است .



شبه کد در صفحه 258- فراخوانی تابع برای مسئله ی 4 وزیر

queens(1,4)

❋ فرض : برنامه مسئله n وزیر را با زبان پیاده سازی و اجرا نمایید .

مسئله حاصل جمع زیرمجموعه ها :

صورت مسئله: ورودی:n عدد صحیح مثبت مقادیر w1,w2,w3,…,wn

خروجی:(xi (x1,x2,…,xn یا 0 یا 1 است

مثال :

w1=5 , w2=6 , w3=15 , w4=11 , w5=16

W=21

W1+w2+w3=21

W1+w2=21

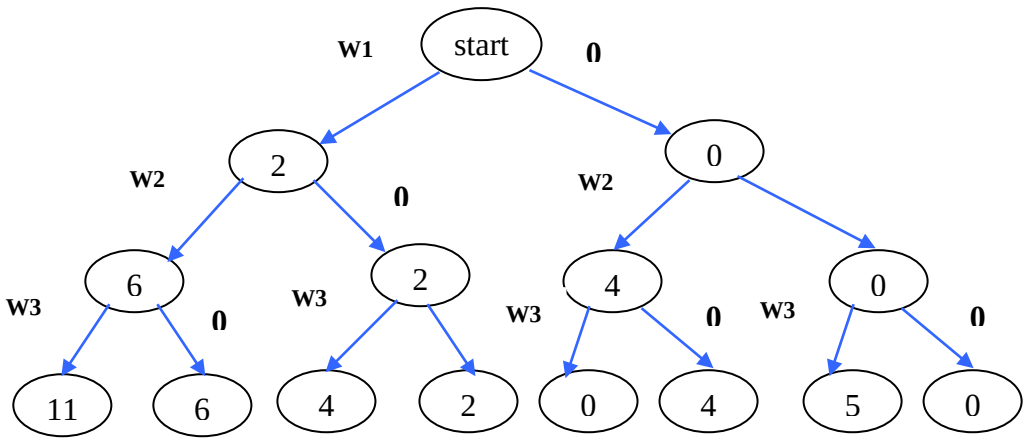
W3+w4=21

مجموعه جواب {w3,w4} {w1,w2} {W1,w2,w3}

مثال دیگر:

w=6

w1=2 , w2=4 , w3=5 → بهتر است ابتدا بصورت صعودی بچینیم



(x1,x2,x3)
(1,1,0)

1*w1+1*w2+0*w3=
1*2+1*4+0*5=6

بررسی امید بخش بودن یک گره:

مرتب سازی وزن i به ترتیب نزولی (برای اینکه کمترین وزن را تا حد ممکن انتخاب کنیم وبا احتمال کمتری به این سمت برسیم.)

اضافه کردن گره پايينی از مرز مجاور

بررسی گره در سطح i

۱. $weight+w_{i+1}>w$

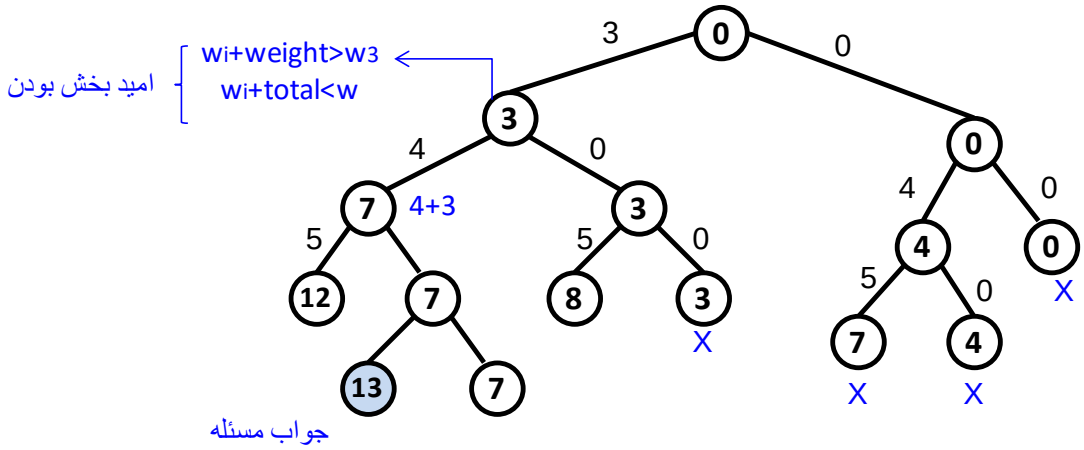
2. $weight+total<w$

هر چقدر پايين تر برويم به وزن مطلوب نهايي نمي رسيم

ردیف	پارامتر	
۱	w	مجموع وزن نهایی
۲	weight	مجمو وزن ها تا این سطح از درخت حالت
۳	Total	جمع اجزایی که هنوز اضافه نشده اند

مثال: درخت فضای حالت هرس شده برای مسئله

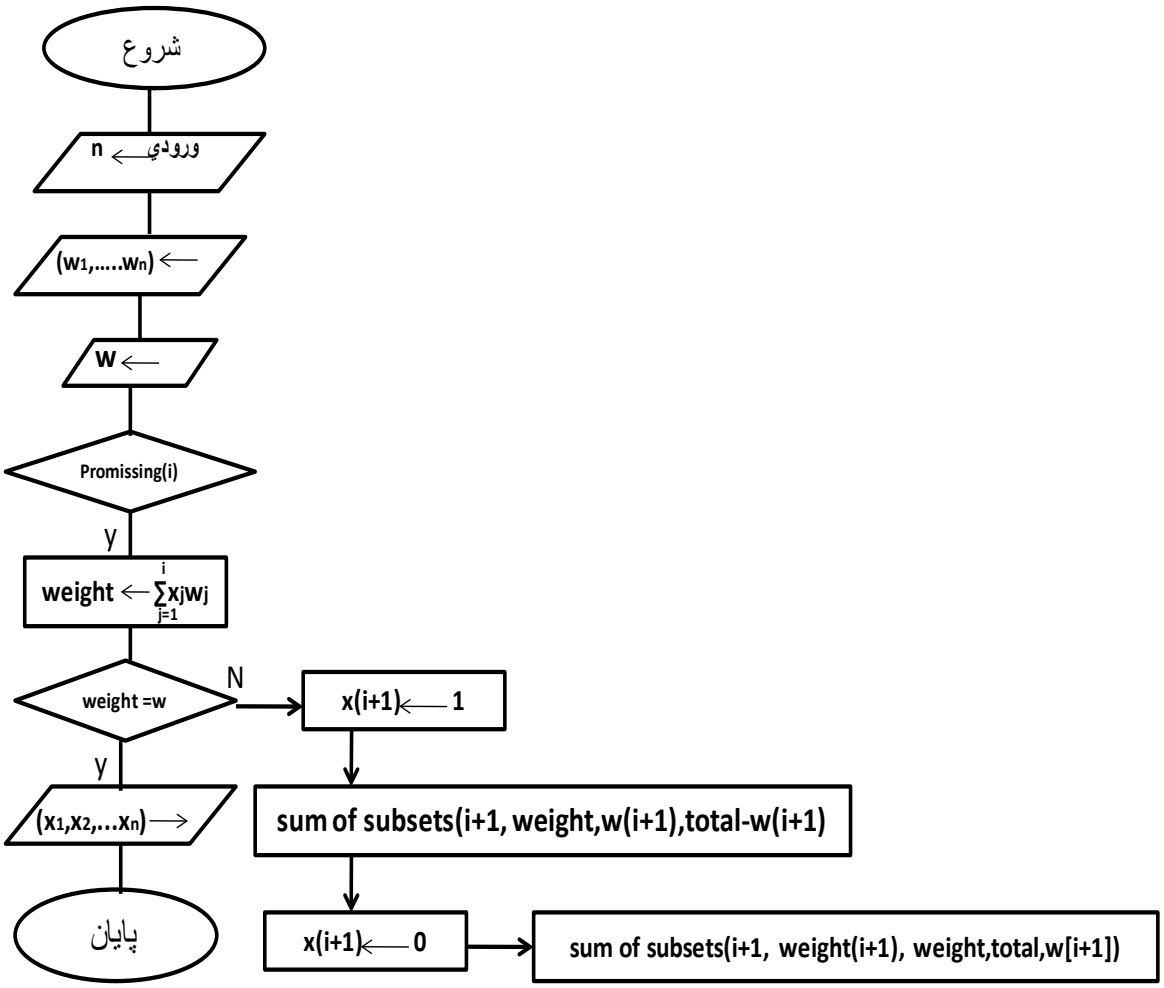
$$\begin{cases} w_1=3 , w_2=4 , w_3=8 , w_4=6 \\ w=13 \end{cases}$$



*موارد X غیر امیدبخش هستند.

الگوریتم مسئله جمع زیر مجموعه ها sum of subsets(i, w, total)

اجرا با دستور sum of subsets(0,0,total) :



الگوریتم امید بخش بودن:

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

پچیدگی زمانی الگوریتم جمع زیر مجموعه ها:

Sum of subsets (0,0,total) ⟶

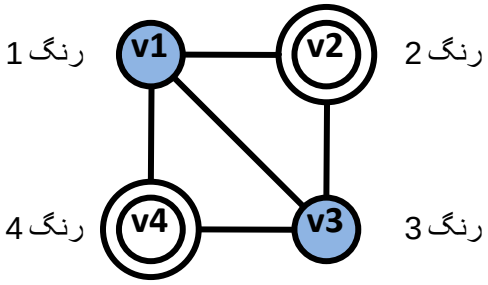
$1+2+2^2+ \ldots + 2^n = 2^{n+1} -1$

اولین فراخوانی تعداد گره های هرس شده

$1+a+a^2 +\ldots+a^n = \frac{a^{n+1} -1}{a-1}$ ⟶ فرمول جمع سري هندسي يا توافقي

رنگ آمیزی گراف:

مسئله: یافتن همه راه حل های ممکن برای رنگ آمیزی رئوس یک گراف بدون جهت با استفاده از حداکثر m رنگ متفاوت به طوری که هیچ دو راس مجاور هم رنگ نباشند

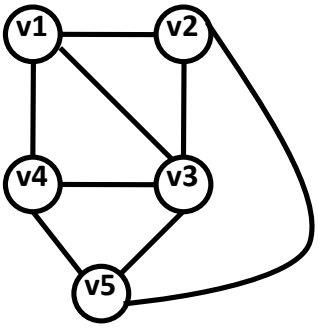


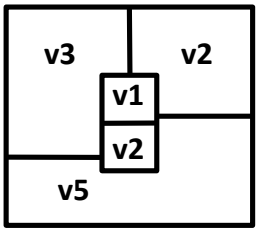
گرافي كه با دو رنگ قابل رنگ آميزي نيست

(حل مسئله با سه رنگ)

کاربرد رنگ آمیزی نقشه:

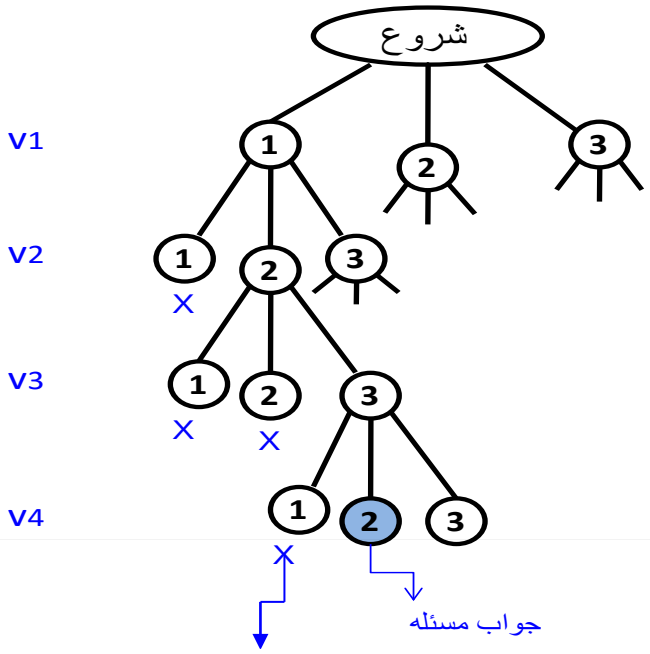
گراف مسطح: گرافی که بتوان آن را در صفحه رسم کرد به طوری که هیچ دو یالی یکدیگر را قطع نکنند.



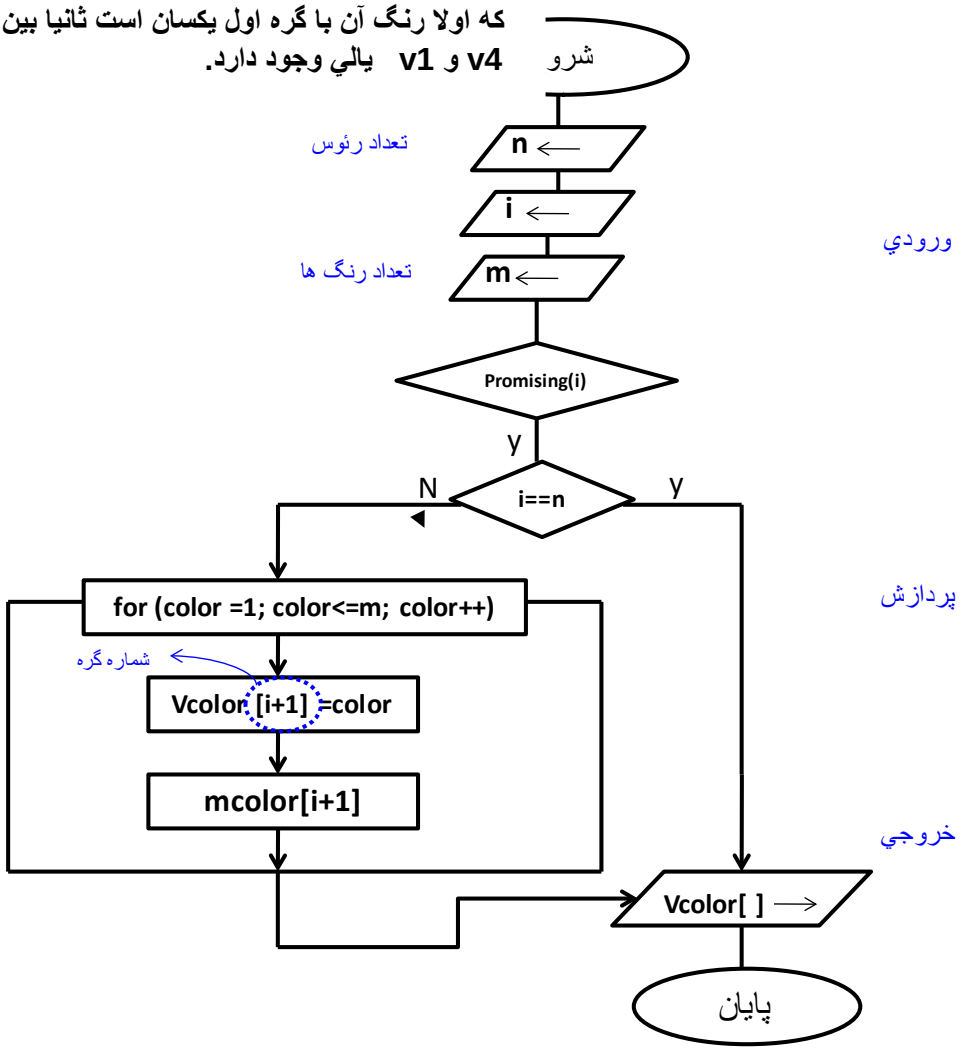


يك نقشه وگراف مسطح هم ارز آن

رنگ آمیزی گراف با سه رنگ:



به طور مثال دلیل این که این گره غیر امیدبخش است این است که اولا رنگ آن با گره اول یکسان است ثانیا بین دو راس v4 و v1 یالي وجود دارد.



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

رنگ دو راس يكي باشد

iوj بين دو راس ارتباط باشد

پیچیدگی زمانی:

فراخوانی تابع در بالاترین سطح vcolor(0)

تعداد گره ها در درخت فضای حالت:

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

مسئله دور هامیلتونی:

مسیری که از هر راس فقط یک بارعبور می کند وبه راس شروع برمی گردد

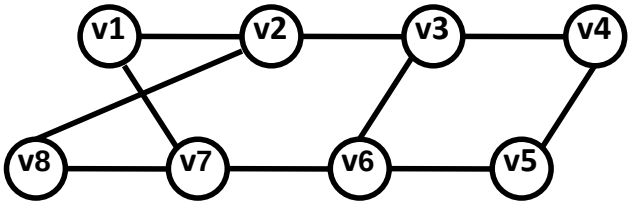
یادآوری:مسئله تعیین کوتاهترین تور با n=20

- روش برنامه نویسی پویا: $2^{n-3}\,T(n)=(n-1)(n-2)$
- روش کورکورانه: $(n-1)!$

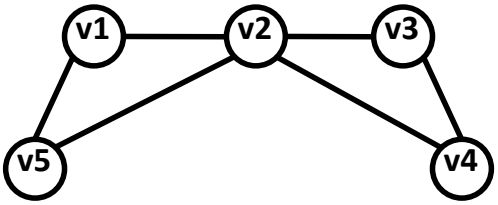
n=40	روش برنامه ريزي پويا	6.46 سال

*مسئله یافتن هر توری در گراف(بدون توجه به وزن تور) را مسئله دورهای هامیلتونی می نامند

الف) داراي دور هاميلتوني



ب) داراي دور هاميلتوني



*تمرین:برنامه مسیر هامیلتونی را به طور کامل پیاده سازی کنید.

مسئله کوله پشتی صفر ویک:

- گره غیر امید بخش weight>=w
- مرتب سازی قطعات بر حسب pi/ iwi به صورت غیر نزولی
- تعیین حد بالا برای سود قابل حصول از گسترش دادن گره(bound)
- profit:حاصل جمع ارزش قطعاتی که تا ان گره در نظر گرفته شده اند
- weight:حاصل جمع اوازن آن قطعات
- مقدار اولیه متغیر bound را برابر profitقرارمی دهیم.
- مقدار اولیه متغییر total weight را برابر weight قرار می دهیم.
- هر بار به روش حریصانه یک قطعه را برداشته وارزش آن را به bound و وزنش را به total weight اضافه می کنیم تا به قطعه ای برسیم که اگر وزن آن را به مجموع وزن های قبلی اضافه کنیم total weight ازw بیشتر شود.
- max profit:مقدار بهره در بهترین حالتی که تا کنون پیدا شده است.

فرض کنید گره در سطح i باشد و گره در سطح k گرهی باشد که حاصل جمع اوزان را از w بیشتر کند در این صورت:

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

وزن قطعه kام باعث مي شود جمع از حد مجاز بيشتر شود

$$\text{bound}=\underbrace{(\text{profit}+\sum_{j=i+1}^{k-1} p_j)}_{\text{بهره حاصل از k-1 قطعه نخست (که مطمئنیم مجموع وزنها از k بیشتر نیست)}}+\underbrace{(w-\text{total weight})}_{\text{ظرفيت باقي مانده براي قطعه لازم}}*\underbrace{p_k/w_k}_{\text{بهره واحد وزن براي قطعه kام}}$$

❊گره غیر امید بخش: bound<=maxprofit (ادامه دادن آن بی مورد می باشد)

مثال: 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 $

max profit=0

profit=0

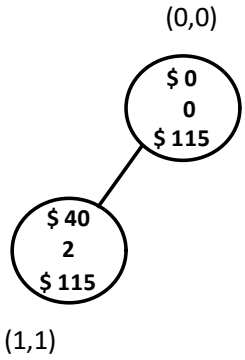
weight=0

$$\text{totalweight}=\text{weight}+\sum_{j=0+1}^{3-1} w_j=0+(2+5)=7$$

$$\text{bound}=(\text{profit}+\sum_{0+1}^{3-1} p_j)+(w-\text{totalweight})*p_3/w_3$$

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

❊گره امید بخش است چون وزنش کمتراز w=16 وحدش بزرگتر از maxprofit است.



Profit =$0+$40=$40>maxprofit=0 → maxprofit=$40
weight =0+2=2

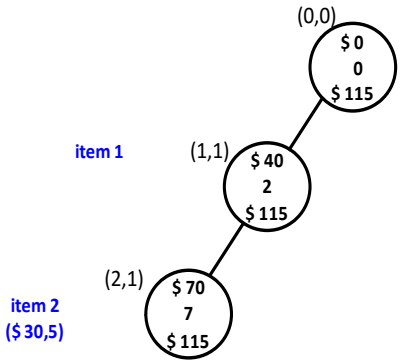
$$\textbf{totalweight=weight}+\sum_{j=i+1}^{3-1} w_j=2+5=7$$

$$\text{bound} =\text{profit}+\sum_{j=i+1}^{3-1} p_j +(w-\text{totalweight})* p_3/ w_3$$

=$0+$40+(+($30))+(16-7) * $50/10=$115

❊ گره امید بخش است چون وزنش کمتر از w=16 وحدش بزرگتر از maxprofit است.

❊ تمرین: این فرمول بندی را در فایل Excel محاسبه کنید.

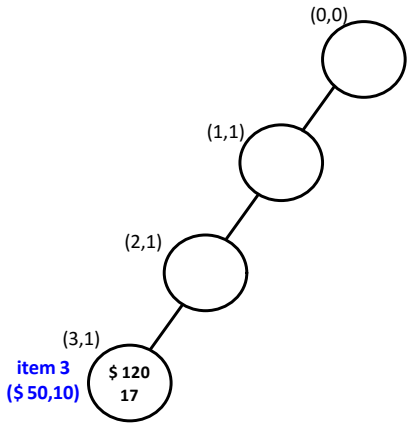


Profit =$40+$30=$70>maxprofit → maxprofit=$70
weight =2+5=7
maxprofit=$70

$$\textbf{totalweight=weight}+\sum_{j=i+1}^{3-1} w_j=7$$

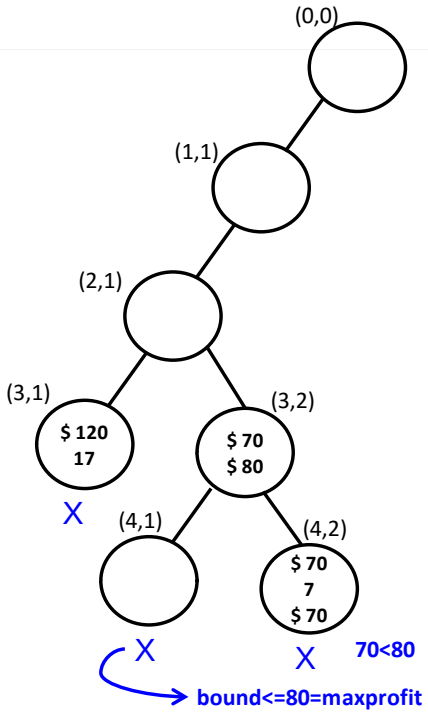
bound =$70+(16-7) * $50/10=$115

❊ گره امید بخش است چون وزنش کمتر از w=16 وحدش بزرگتر از maxprofit=70 است.



Profit =$70+$50=$120
weight =7+10=17>16 غير اميد بخش
maxprofit=$70 تغيير نمي كند

* چون وزن بيشتر از حد مجاز است نيازي به محاسبه **bound** نيست



Profit =$70
weight =7
maxprofit=$70

bound =profit+$\sum_{i=3+1}^{5-1}$ **pj=$70+10=$ 80**

فصل ششم

انشعاب و تحدید

انشعاب و تحدید: Branch & Bound

تکنیک B&B یک تکنیک برنامه نویسی است که در آن عملکرد تکنیک Back tracking برای برخی از مسائل می تواند بهبود یابد: با این مفهوم که با یک مکمل اولیه از هزینه یک راه حل به دنبال یک راه حل بهتر هستیم. برای این منظور در هر کجای مسیراز راه حل بعدی' اگر میزان هزینه بیشتر از هزینه اولیه باشد آن مسیر حذف می شود.(جهت تعیین یک تخمین اولیه می توان از روش حریصانه استفاده کرد)

ویژگی های کلی:

۱.ما را به روش خاصی از پیمایش درختی محدود نمی کند

۲.تنها برای مسائل بهینه سازی به کار می رود

۳.جستجو بر اساس اول وبهترین انجام می پذیرد.

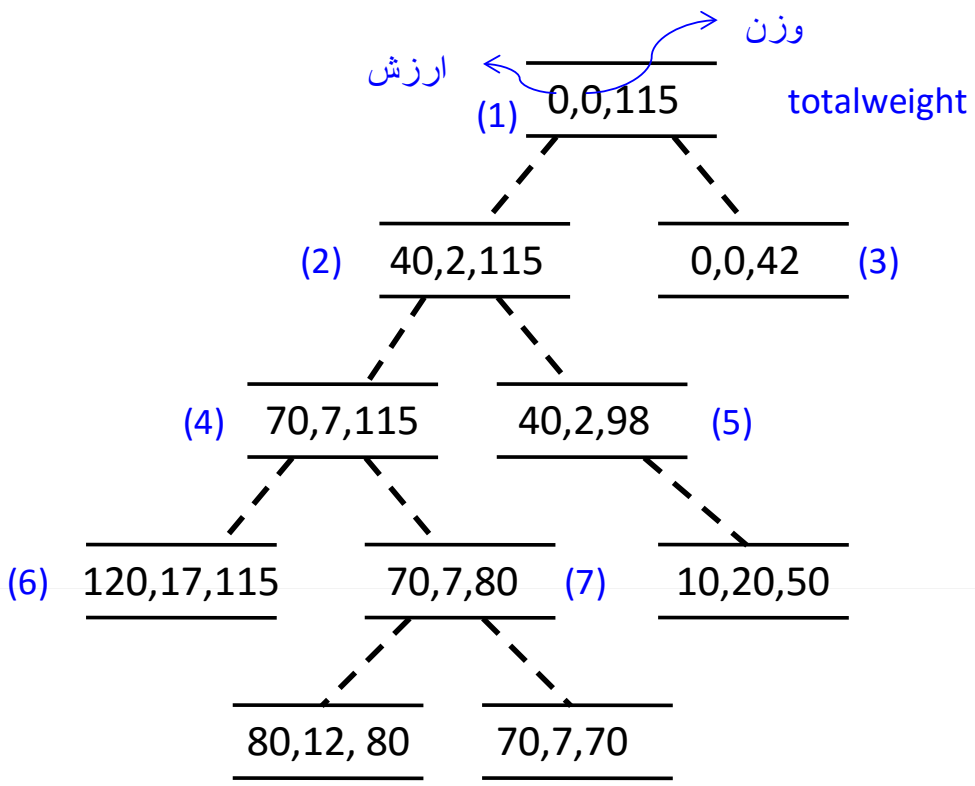
توجه: در روش شاخه وحد مر تبه زمانی در بدترین حالت تغییر نمی کند اما در عمل تعداد عملیات مورد نیاز کمتر خواهد بود.

مثال: کوله پشتی صفرو یک

i	pi	wi	pi/ wi
1	20 $	2	10 $
2	30 $	5	6 $
3	50 $	10	5 $
4	10 $	5	2 $

الف) کوله پشتی صفرو یک با هرس شاخه وحد وبه صورت سطح اول

ب)کوله پشتی با هرس شاخه وحد واول بهترین در این روش در هر لحظه از میان نودهای باز فقط آن را که بهترین حالت است بسط می دهیم ابتدا بررسی می کنیم کدام یک از نودها حد را رعایت کرده (دراین مثال مقدار Bound آنها از Max profit بیشتر است) و سپس از بین آنها max را انتخاب می کنیم.



❋ابزار soluer در Excel: در بخش soluer, tools/option/ add in را تیک دار می کنیم.

❋macro rs در Excel: می توان با آن برنامه ها را پیاده سازی کرد- برای تصمیم گیری

❋ سیستمDss: Decision support system : سیستم تصمیم ساز می سازد

.....what if: دور نمایی از کاری که می خواهیم انجام بدهیم در اختیارمان قرار می دهد.

❋ این روش برای مسئله کوله پشتی هم عرضی است وهم عمقی

فروشنده دوره گرد:

تابع ارزیابی ' کمترین هزینه ممکن برای یک گردش برابر می باشد با مجموع کمترین هزینه ورود وخروج هر گره در آن توروتقسیم حاصل جمع بر۲

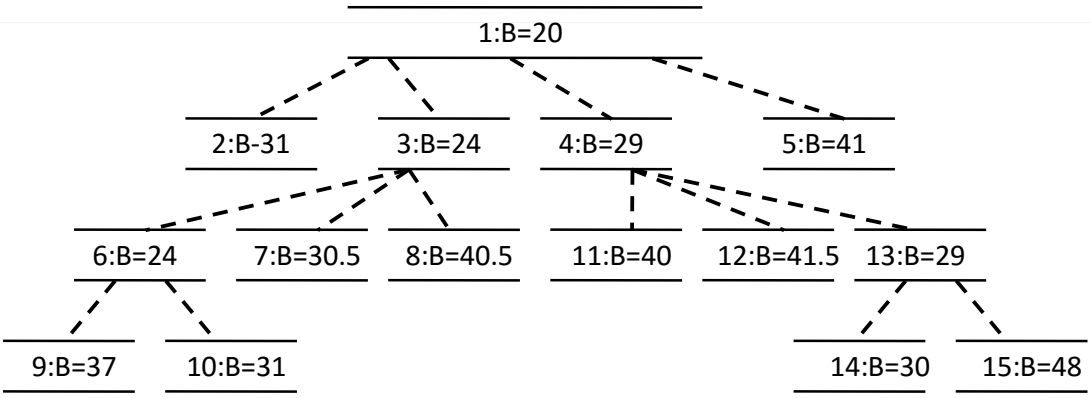
روش سطرو ستونی:

$$
\begin{array}{c|ccccc}
 & 1 & 2 & 3 & 4 & 5 \\
\hline
1 & 0 & 14 & 4 & 10 & 20 \\
2 & 14 & 0 & 7 & 8 & 7 \\
3 & 4 & 5 & 0 & 7 & 16 \\
4 & 11 & 7 & 9 & 0 & 2 \\
5 & 18 & 7 & 17 & 4 & 0
\end{array}
$$

$$B=\sum_{i=1}^{n} (\min(\text{سطر}i\text{ام})+\min(\text{ستون}i\text{ام}))/2$$

$$=(4+4+7+5+4+4+4+2+2+4)/2=20$$

❋روش دیگر برای محاسبهB: 14+7+2+4+2+4=31



محاسبه هزینه تور:

۱.تا لبه(1,2) به صورت قطعی انتخاب شده است از روش ورودی به گره ۳ وخروجی از گره ۱ برابر ۱۴ می باشد.

۲.خروجی از گره ۲ نمی تواند ورودی به گره ۱ باشد

۳.ورودی به هیچ گره ای به جز ۲ نمی تواند از گره ۱ باشد

۴.خروجی از هیچ گره ی دیگر نباید به گره ۲ باشد

❋ در صورتی که (۱,۵) را انتخاب می کنیم دیگر نمی توانیم (۵,۱) را انتخاب کنیم.

فصل هفتم

مسائل رام نشدنی

مسائل تصمیم گیری و دسته بندی آنها

کلاس p : مجموعه مسائل تصمیم گیریکه برای آنها یک الگوریتم قطعی با زمان چند جمله ای وجود دارد.

کلاس NP: (Nondeterministic Polynomial): مجموعه مسائل تصمیم گیری که برای آنها یک الگوریتم غیر قطعی با زمان چند جمله ای وجود دارد.

مثال از p	مثال از Np
۱.تشخیص مرتب بودن یک آرایه ۲.تشخیص این که یک رشته مفروض شامل زیر رشته خاص هست یا نه	Sum of subset

نکته : هر مسئله p متعلق به Np نیز می باشد به عبارتی p<=Np زیرا اگربرای مسئله ای الگوریتم قطعی چند جمله ای وجود داشته باشد می توان همان الگوریتم را غیر قطعی نیز در نظر گرفت.

❋هر مسئله قطعی نیز **(deterministic)** به صورت غیر قطعی**(nondeterministic)** وجود دارد.

❋ چند جمله ای ها رام شدنی هستند.

نکات تستی:

رام شدنی: از مرتبه ی چند جمله ای

رام نشدنی : از مرتبه نمایی ..... 2,n! (مسئله کوله پشتی ' مسئله فروشنده دوره گرد)

* Np complete:رام نشدنی هستند ولی اثبات نشده که برایشان الگوریتم چند جمله ای وجود دارد.

* Np Hard: رام نشدنی هستند وثابت شده که الگوریتم چند جمله ای برایشان وجود ندارد.

* اکثر مسائل Np complete هستند.

* اگر یک راه چند جمله ای برای یکی از Np completeها پیدا شود برای بقیه نیز پیدا می شود.

مسئله زمان بندی:

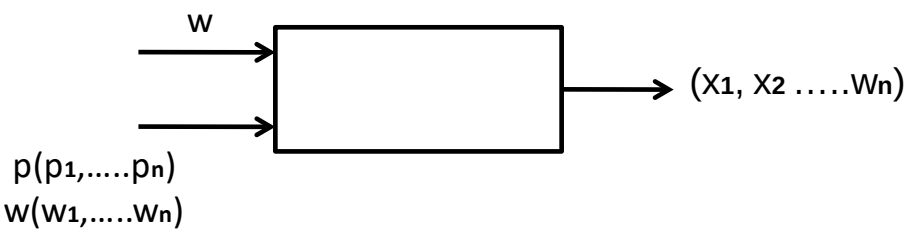
دو گونه زمان بندی داریم:

۱. کمینه سازی کل زمان برای انتظار کشیدن وسرویس دهی کارها

زمان بودن در سیستم time in the system

مثال: آرایشگاه (اصلاح سر)

کوله پشتی با برنامه نویسی پویا:



ارزش قطعات از اول تا i-1

$$p[i][w]=\begin{cases} \text{maximum}\,(p[i-1]\,[w],\ p_i+p[i-1]\,[w-w_i]\,) & w_i<=w \\ p[i-1][w] & w_i>w \end{cases}$$

ارزش قطعه i ام

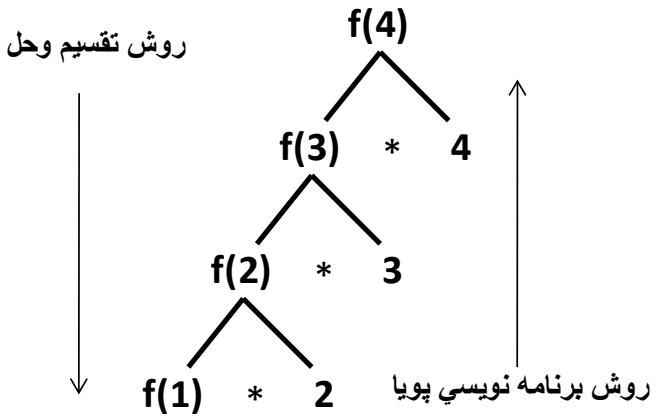
ارزش با این فرض
که n-1 انتخاب
یعني

سود بهینه حاصل از انتخاب i قطعه اول به
شرطي که از w (وزن کل) بیشتر نشود

❊ اگر Wi (وزن قطعه i ام) از W بیشتر باشد' دیگر هیچ شی ای به داخل کوله اضافه نمی شود

❊ اگر در نهایت وزن از W بیشتر شود' p[i-1] [w] ارزش است ودر غیر این صورت قطعه مورد نظر اضافه نمی شود.

f(n)=f(n-1)*n



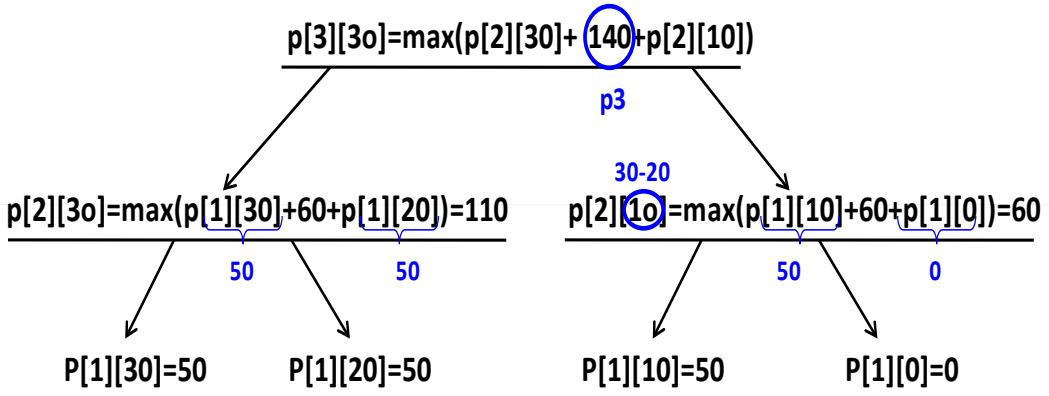
برنامه نویسي
پویا

```
F[1]=1
for (i=1; i<=10; i++)
    f[i]=i * f[i-1]
```

روش تقسیم
وحل

```
f(n)
return(f [n-1] * f [n] )
```

$$p[n][w]=\begin{cases} \text{maximum}\,(p[n-1]\,[w],\ p_n+p[n-1]\,[w-w_n]\,) & w_n<=w \\ p[n-1][w] & w_n>w \end{cases}$$



❊ اگر وزن صفر باشد ارزش نیز صفر است.

❊ ابتدا درخت را تشکیل می دهیم سپس از پایین به بالا محاسبات را انجام می دهیم.

تمرین : مسئله کوله پشتی به روش برنامه نویسی پویا را در Excel حل کنید.

نمایی از حل مسئله:

همان w است

g	-	0	2	y	.....	n-1	M
0							
1							
. . .							
i							
n-1							
n							

جواب

تابع $g(y)=\max(g_{i+1}(y)+p_{i+1}+ g_{i+1}(y,w_{i+1})$

$g_i(y)$: بیشترین سود حاصل از حل مسئله برای جسم $i+1$ ام تا n ام به شرطی که گنجایش کوله پشتی y باشد.

$g_0(m)$:بیشترین سود حاصل از حل مسئله برای کل اجسام به شرطی که گنجایش کوله پشتی m باشد.

شرایط مرزی:

$g_i(y)=0$:(از جسم $n+1$ به بعد جسمی وجود ندارد که سودی داشته باشد)

$y<0$:(گنجایش کوله پشتی منفی باشد یعنی بیشتر از حد کوله پشتی جسم داریم

ودر این صورت $g_i(y)$ حالت ضرر دارد $g_i(y)=-\infty$