

بسم الله الرحمن الرحيم

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

www.algorithm.design.blogfa.com

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 جزئی نگری در شناخت (reductionism)
- دیدگاه کل نگر در شناخت (holistic)
- تابع جستجوی دودویی به صورت بازگشتی (Binary search)
و پیچیدگی زمانی آن
- تابع جستجوی دودویی به صورت غیر بازگشتی (Binary search)
و پیچیدگی زمانی آن
- مرتب سازی ادغامی (Merge sort) و پیچیدگی زمانی آن
- مرتب سازی سریع (Quick sort) و پیچیدگی زمانی آن
- ضرب ماتریس ها به روش استراسن (Strassen) و پیچیدگی 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
- فروشندگی دوره گرد
- محاسبه هزینه تور

فصل هفتم: مسائل رام نشدنی

- مسائل تصمیم گیری و دسته بندی آن ها
- کلاس p
- کلاس np
- 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 در **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 تنهایی 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 - آرایه)

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 داده 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 مسلماً 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.

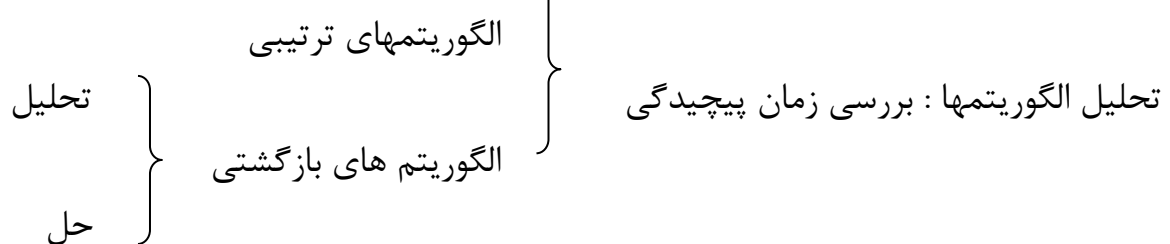
۴- ترکیب داده های ورودی : بسته به شرایط (از بدترین حالت - متوسط - بهترین حالت)

۵- پیچیدگی زمانی الگوریتم: تابعی از اندازه های مسئله

۶- پارامترهای دارای تاثیر ثابت در زمان اجرا

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

Road map : نقشه کلی درس طراحی الگوریتم ها



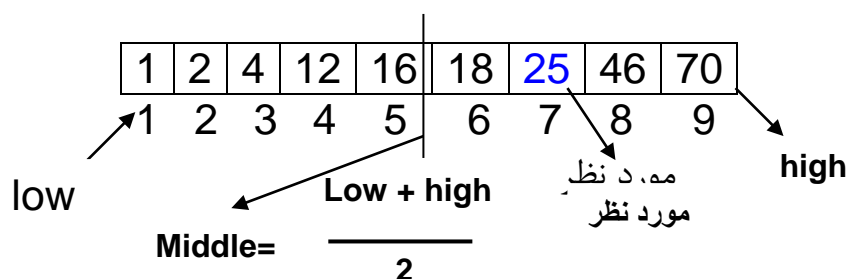
* توجه شود که ۳ فصل اول طراحی الگوریتم ها مربوط به فصل اول و آخر ساختمان داده ها است.

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) →

core هسته

Else

Bin search (middle + 1 , high) ; }

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

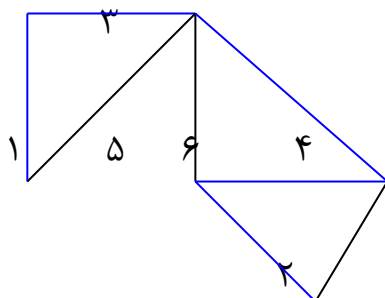
یادآوری : برای توابع بازگشتی از پشته استفاده میشود

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

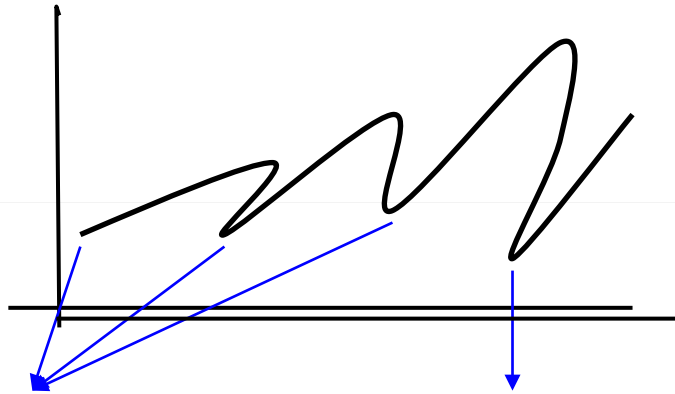
یک گراف (یعنی نباید حلقه داشته باشیم)



$$\text{sum} = 3 + 1 + 4 + 2 + 1 = 11$$

تمام رئوس پوشش داده میشود و 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.



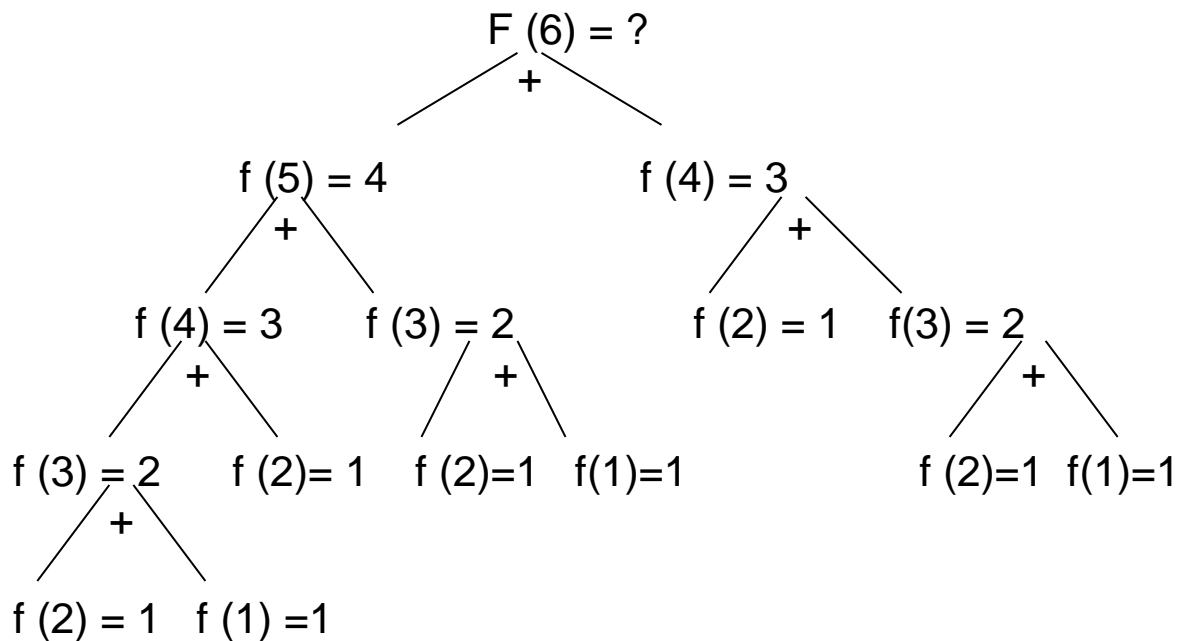
Min محلی

min مطلق

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

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

$$F(2) = f(1) = 1$$



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 4×4 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) $C_1 \quad X++ ;$

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

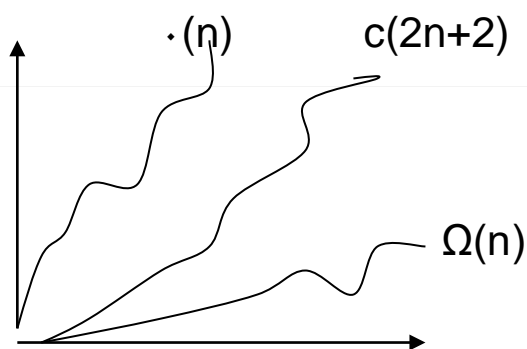
4	C_4	$n-1$	$C_4(n-1)$
5	C_5	1	C_5

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

$$c = \max(C_1, \dots, C_5) = \max(C_i)$$

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

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



نماد **Big - oh** حد بالای $t(n)$

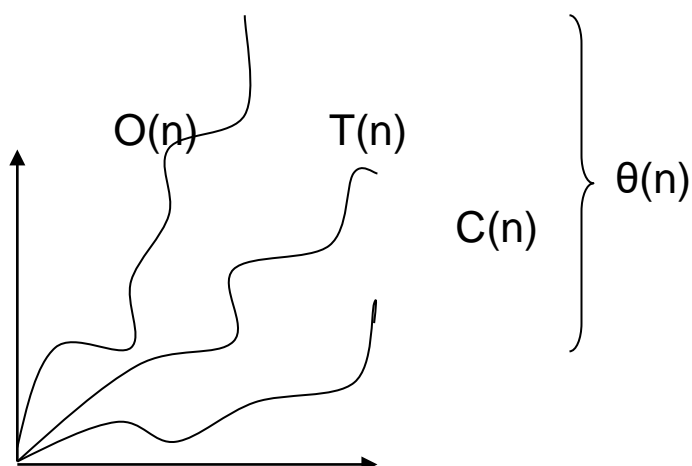
$$T(n) = C(2n+2) \quad t(n) = c(2n+2) < 2Cn + 2Cn \quad T(n) \in o(n)$$

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 $n < k$ 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:

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

رابطه کلی ($a, b > 1$)

$$G = \text{درجه رشد} = g(n) = n^{\log_a b}$$

$$F = \text{درجه رشد} = F(n)$$

$$f(n) \in \theta(g(n)) : F > G \quad (\text{الف})$$

$$T(n) \in \theta(g(n)) : F < G \quad (\text{ب})$$

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

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

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

$$C(n=0)$$

$$\left. \begin{array}{l} 1. \text{ همگن } C=0 \\ 2. \text{ غیر همگن } C \neq 0 \end{array} \right\} \leftarrow \text{تغییر متغیر}$$

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

$$d_n = \frac{c_n}{b_1 b_2 \dots b_n} \rightarrow t(n) = (b_1 b_2 \dots b_n) (T(0) + b_1 \dots b_2)$$

(۲) رابطه بازگشتی مرتبه دوم:

$$\left. \begin{array}{l} At(n-1) + bT(n-2) \quad (n > 1) \end{array} \right\}$$

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

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

حل میکنیم $\left\{ \begin{matrix} X_1 \\ X_2 \end{matrix} \right\} \rightarrow \left\{ \begin{matrix} X_1, X_2 \\ X_1, X_2 \end{matrix} \right\}$ متمایز X_1, X_2

$\rightarrow T(n) = c_1 x_1^2 + c_2 x_2^n$ متمایز میکنیم: C_1, C_2 را به کمک شرایط اولیه تعیین میکنیم:

$\rightarrow T(n) = c_1 x_1^n + c_2 n x_2^n$ یکی هستند X_1 و X_2

(۳) رابطه مرتبه دوم غیر همگن

$$T(n) = \underbrace{aT(n-1)}_{Y'} + \underbrace{bT(n-2)}_{Y''} + c F(n)$$

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

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

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

معادله دیفرانسیل که مشتقات y در آن وجود

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

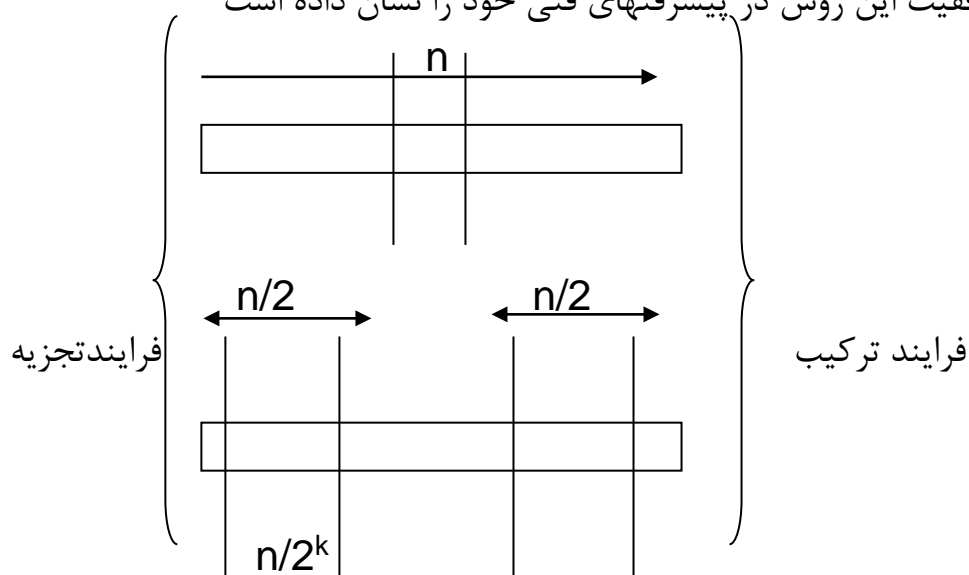
روش تقسیم و حل (Divide & conquer)

روش تقسیم و حل (divid & conquer):

دیدگاه جزئی 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 جزئی 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 عمدتاً مبتنی بر نگرش شهرتی میباشد در ابتدا یک شناخت کلی از مسئله و سیستم حاکم بر آن حاصل میگردد و عمدتاً 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

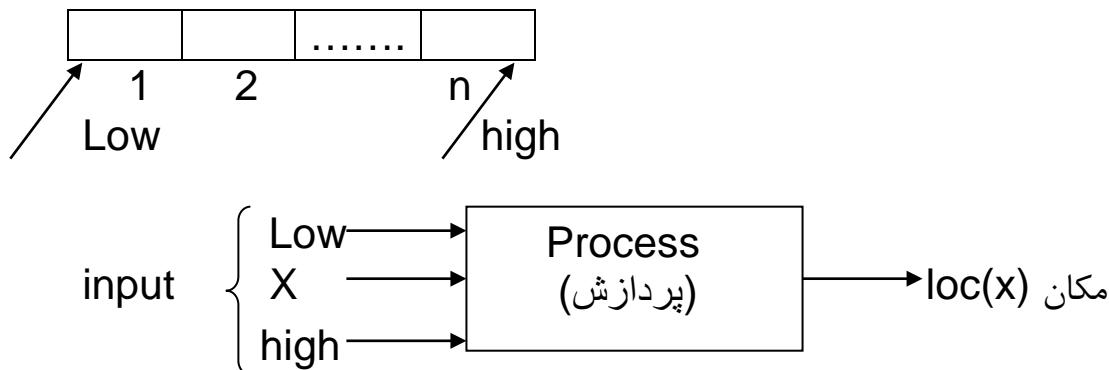
۲. انتخاب یا طراحی ساختار اطلاعاتی

۳. طراحی الگوریتم و فلوچارت

۴. پیاده سازی

۵. خطا زدایی

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

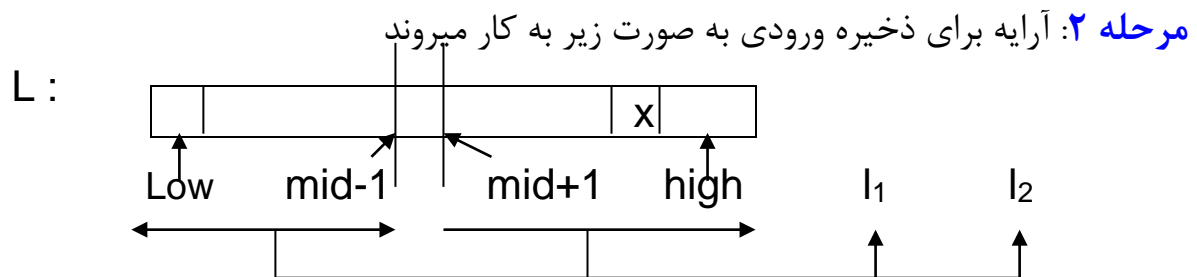


binsaerch (l , low , high , x) output

شکل ۲) شمای کلی الگوریتم

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

خروجی	← loc خروجی
-------	-------------



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

شکل ۳ - ساختار اطلاعاتی مسئله برای ذخیره ورودی

مرحله ۳ ایده کار با روش تقسیم و حل

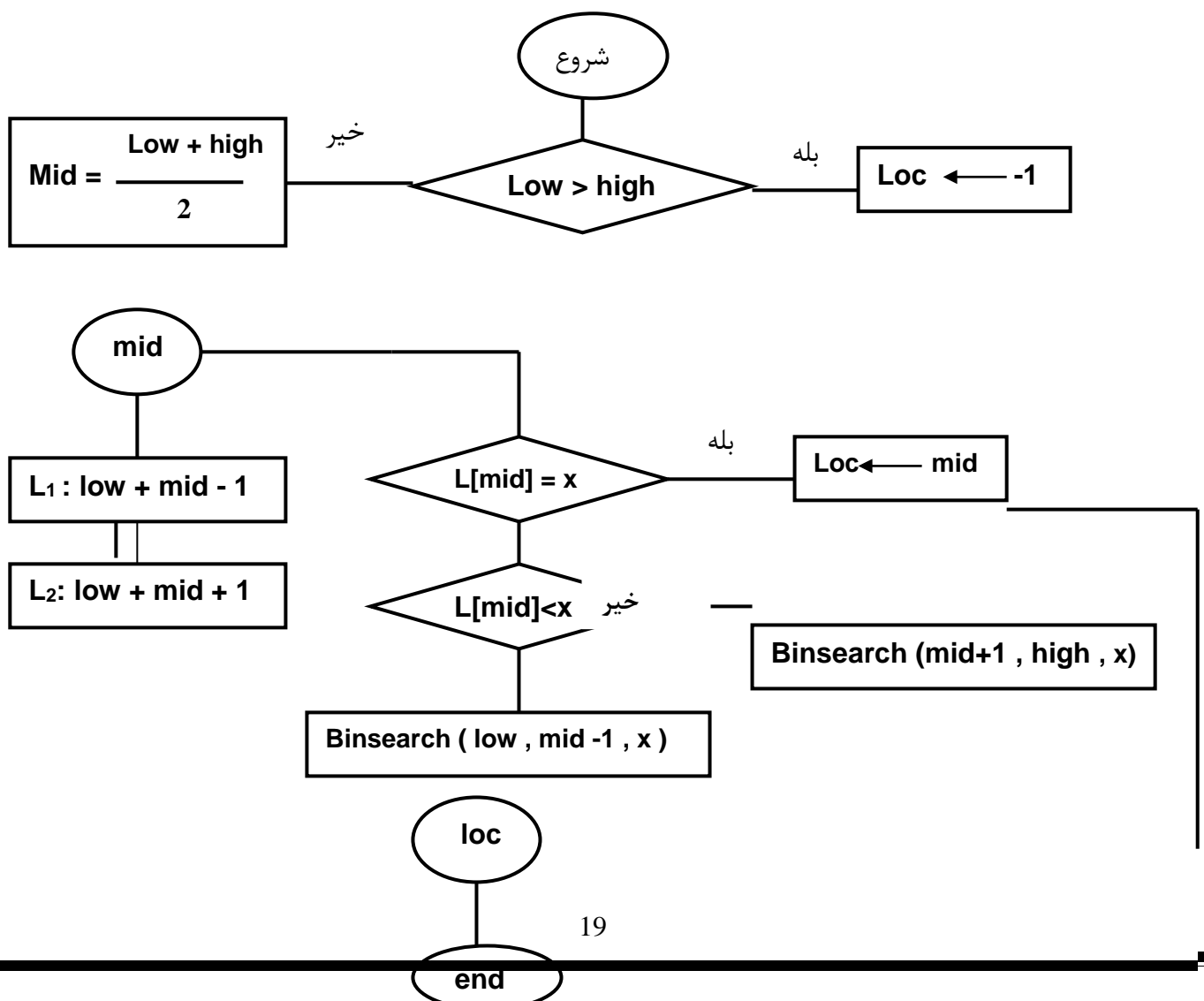
تقسیم ساختار اطلاعاتی به دو قسمت و یافتن مقدار x در یکی از آنها با کمک فراخوانی تابع جستجویه صورت بازگشتی

- 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 ← low ورودی ← $high$ ورودی ← x
پردازش	۲. اگر $low > high$ ← loc ۳. یافتن عنصر وسط آرایه $low + high$ $Mid = \frac{\quad}{2}$ ۴. تقسیم آرایه به دو نیمه با کمک عنصر وسط ۵. اگر عنصر وسط $x = mid$ ← $Loc \leftarrow mid$ و برو 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)	
۸. $loc \leftarrow$ خروجی	خروجی
۹. پایان	

نکته : x و l پارامترهای ثابت در تمام فراخوانی بازگشتی هستند و لذا میتوان به عنوان پارامتر در پیاده سازی از آنها به جهت استفاده مناسب از حافظه صرف نظر کرد در این صورت نیازی به ذخیره آنها در 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>
int binsearch (l, low , high , x )
{ if ( low > high )
    Return ( -1 )
Else {
    Mid = low + high / 2;
    if x== l[mid]
        Return mid ;  $\longrightarrow$  خروجی
    Else if ( x < a[mid]
        Return binsearch ( l,low,mid-1,x)  $\longrightarrow$  T(n/2)
        Else return binsearch ( l,mid+1,high) ;  $\longrightarrow$  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 $loc = \text{binsearch} (l , low , high , x)$
 خروجی $cout \leftarrow loc$

پیچیدگی زمانی مسئله قبل (جستجوی دودویی)

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

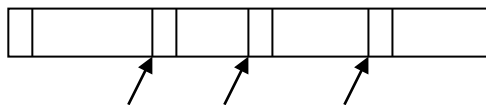
$$T(n/2) + d = T(n/2^2) + 2d = \dots = T(n/2^k) + kd$$

$$n/2^k = 1 \longrightarrow k = \log_2 n$$

$$T(n) = c + d \log_2 n = O(\log n) \quad T(n) = \begin{cases} O(\log n) & \text{جستجوی موفق} \\ \Theta(\log n) & \text{جستجوی ناقص} \end{cases}$$

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 مستقیماً به جواب منتهی شوند میتوان از دیدگاه
 تقسیم و حل به نحوی استفاده کرد



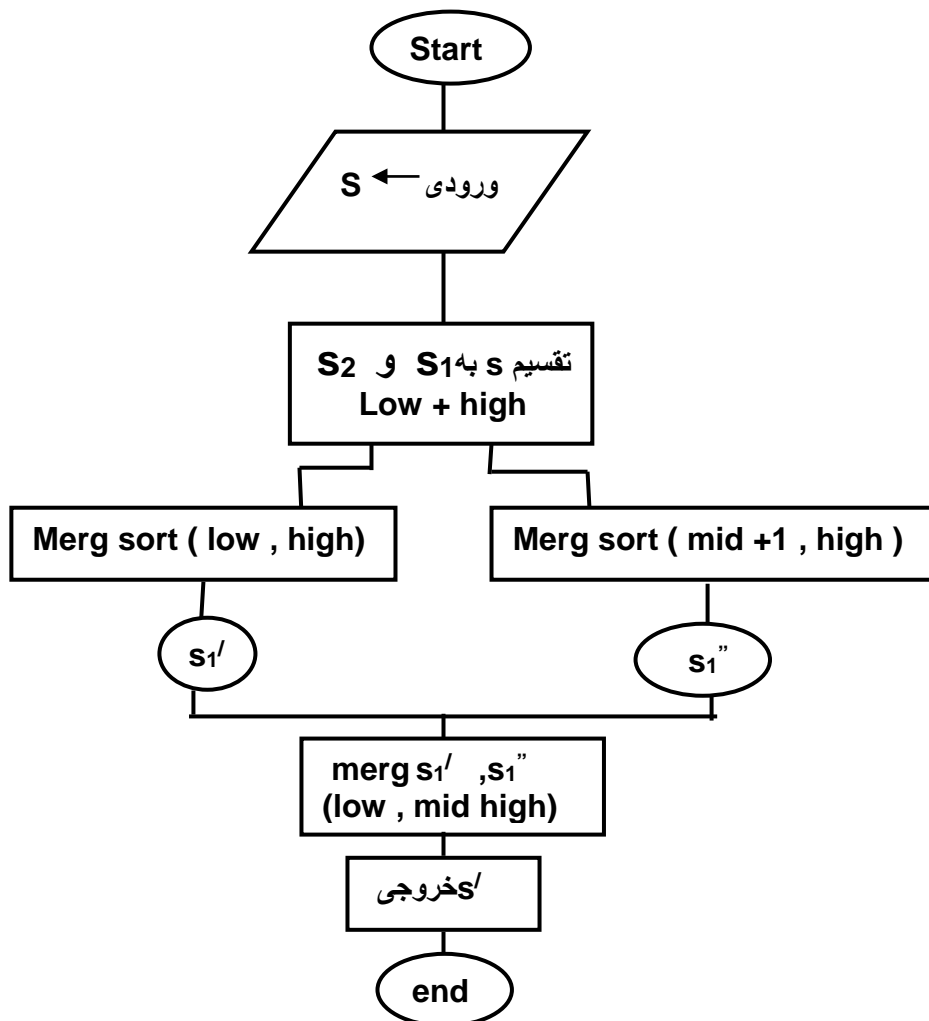
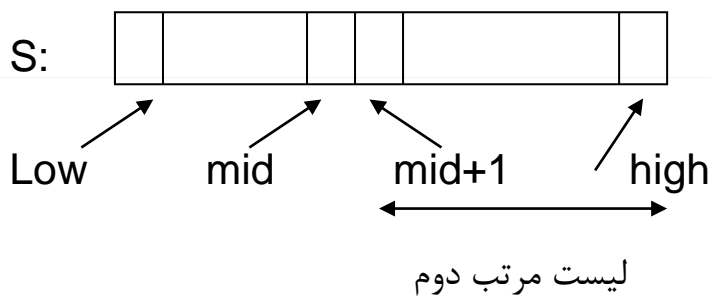
شروع	
ورودی	۱. دریافت ورودی (x , high , low , l)
پردازش	۲. $mid \leftarrow (low + high) / 2$
	۳. اگر $x = l[mid]$ $\leftarrow mid$ low برو به ۸
	۴. اگر $x < l[mid]$ $\leftarrow m-1$ $high$
	۵. برو به ۲
	۶. اگر $x > l[mid]$ $\leftarrow mid - 1$ $high$
خروجی	۷. برو به ۲ (اطلاعات قبلی مورد نیاز نیست لذا پشته ورودی ندارد)
	۸. $loc \leftarrow$ 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 آنها در یک آرایه مرتب



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

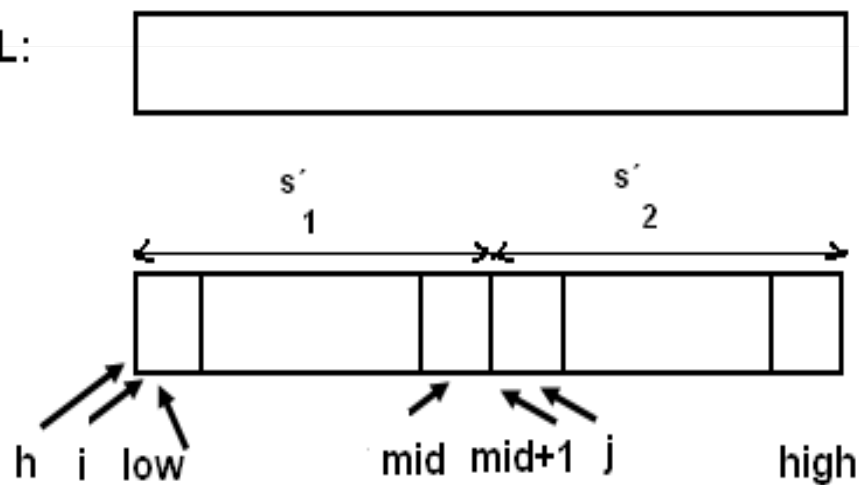
✓ شناخت و تجزیه و تحلیل



✓ ساختار اطلاعاتی مسئله

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

L: آرایه کمکی

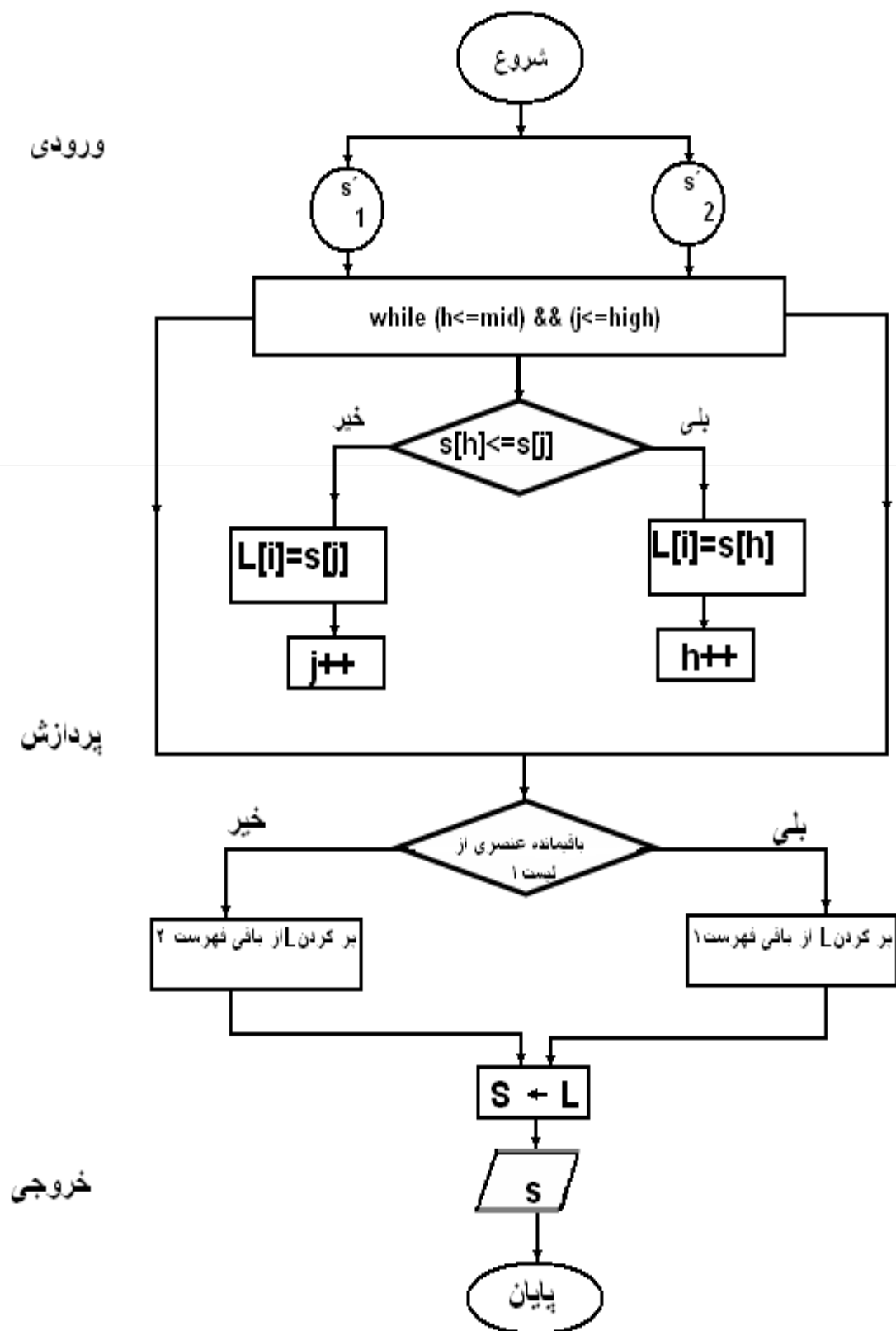


الگوریتم و فلوچارت مسئله:

- 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 L از فهرست اول پرمی شود. در غیر این صورت آرایه L از فهرست دوم پرمی شود.
- اگر عنصری از لیست اول نمانده باشد، آرایه ی کمکی L از عناصر باقی مانده ی فهرست دوم پرمی شود.
- در غیر این صورت عناصر از لیست باقی مانده ی ۱ را در آرایه ی کمکی قرار می دهیم.
- _ اطلاعات آرایه کمکی L را در 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.



پیچیدگی الگوریتم مرتب سازی ادغامی :

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

زمان ترکیب دو زیر رشته مرتب

زمان ۲ بار فراخوانی

حل: با یکی از روش های حل مانند روش تکرار و جایگذاری

$$\begin{aligned} T(n) &= 2T(n/2) + cn \\ &= 2(T(n/2^2) + (n/2)) + cn \\ &= 2(T(n/2^3) + cn/2^2 + cn/2) + cn \\ &= \dots \\ &= 2(T(n/2^k) + cn/2^{k-1} + cn/2^{k-2} + \dots) + cn \\ &= \dots \\ T(n) &\in \theta(n \log n) \end{aligned}$$

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) معمولاً "عنصر اول برای تقسیم به دو

لیست در نظر گرفته می شود.

✓ قرار دادن عناصر کوچکتر از pivot در لیست سمت چپ و بزرگتر از آن در لیست

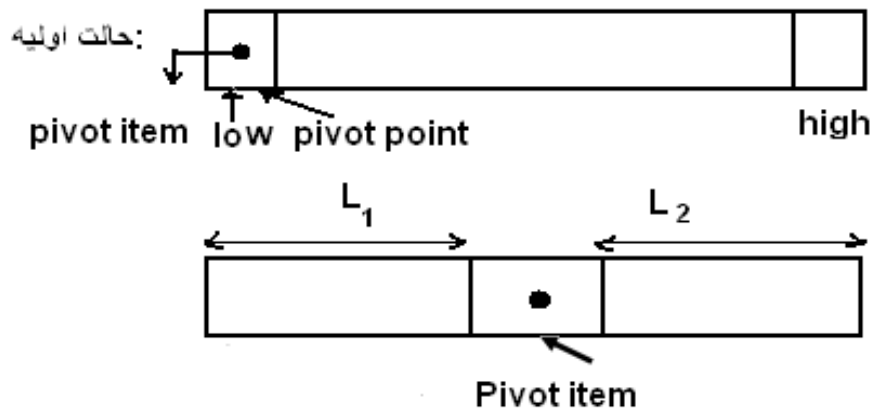
سمت راست و در حقیقت به معنی قرار گرفتن عنصر pivot در جای واقعی خودش.

✓ انجام کارهای ۱ و ۲ 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.



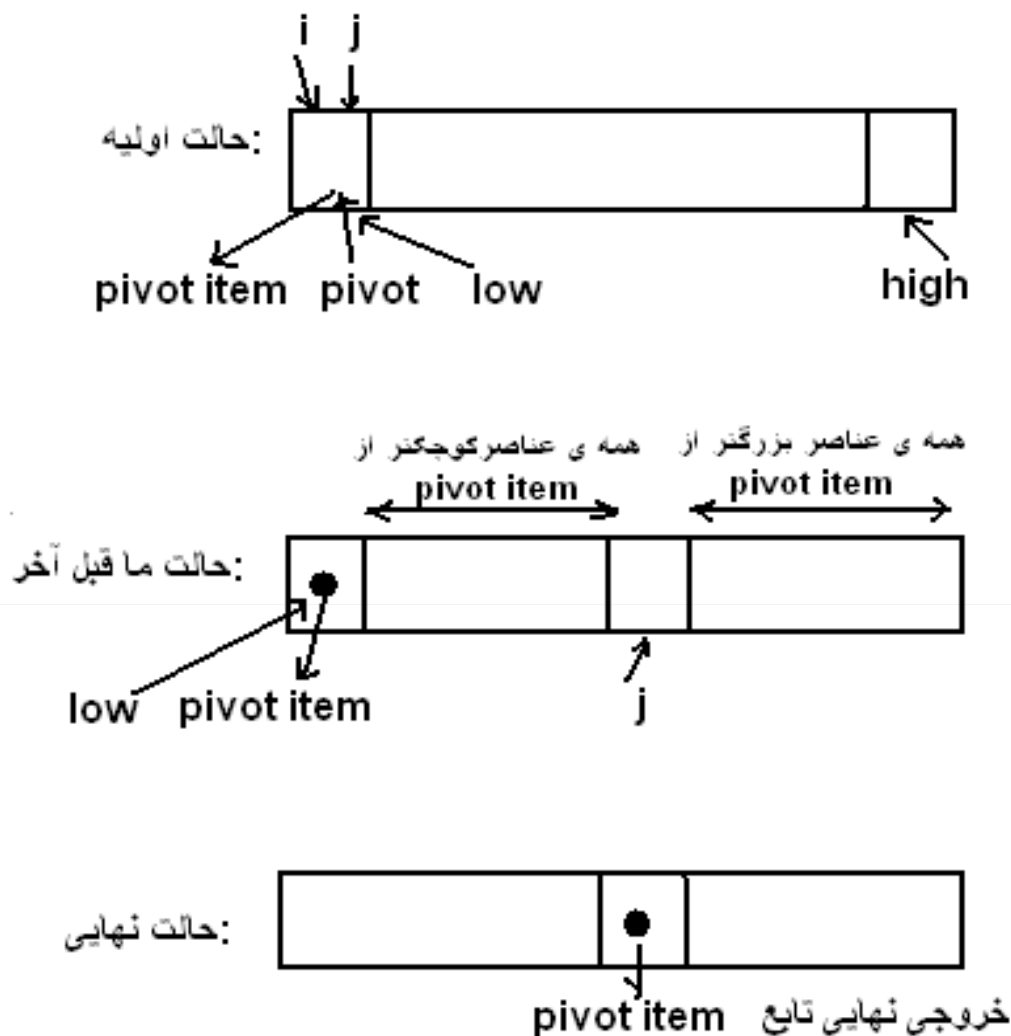
* لیست L الزاما به دو قسمت مساوی تقسیم می شود.

- 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 و هر جا عنصر مورد اشاره کوچکتر از عنصر $pivot$ بود اندیس j یکی به جلو حرکت می کند و در 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 از $pivot$ 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، $1/4$ و... 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 جز عنصر pivot هستند. (به عبارتی داده ها مرتب هستند).

بدترین حالت: $T(n) = T(0) + T(n-1) + (n-1)$

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

پیچیدگی الگوریتم quick sort در حالت متوسط:

در حالت متوسط pivot point همیشه و در هر 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 $1/n$).

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

زمان بخش بندی لیست
تعداد عناصر در سمت
تعداد عناصر در سمت

pivot item چپ
pivot item راست
Partition

pivot point: زمان مرتب کردن سمت چپ

P_{-} مقداری است که توسط تابع بر گردانده می شود.

اثبات:

$$1: nT(n) = (T(0) + T(1) + \dots + T(n-1)) + (T(n-1) + T(n-2) + \dots + T(0)) + n(n-1)$$

$$nT(n) = 2(T(0) + T(1) + \dots + T(n-2) + T(n-1)) + n(n-1) \quad n \rightarrow n-1;$$

$$2: (n-1)T(n-1) = 2(T(0) + T(1) + \dots + T(n-2)) + (n-1)(n-2) \quad 1-2;$$

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

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

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

$$\Rightarrow T_1(n) = \begin{cases} T_1(n-1) + \frac{p(n-1)}{p(n+1)} & \text{if } (n \geq 1) \\ 0 & \text{if } (n < 1) \end{cases}$$

حل رابطه بازگشتی (با استفاده از روش کسرهای جزئی):

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

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

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

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

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

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

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

$$-2 \left(\frac{1}{n} + \frac{1}{n-1} + \dots + \frac{1}{1} \right) + \frac{4}{n+1} + 4 \left(\frac{1}{n} + \dots + \frac{1}{2} + 1 \right) =$$

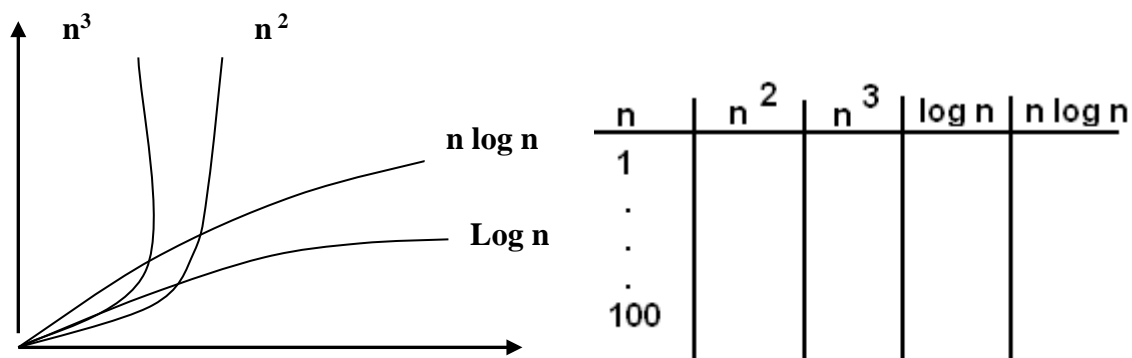
S

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

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

$$\Rightarrow \frac{T(n)}{n+1} \in \Theta \ln(n) \Rightarrow T(0) \leq \Theta(n \log n)$$

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



ضرب ماتریس ها به روش استراسن:

- ضرب دو ماتریس $n \times n$

- 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 ی 4-5 برای محاسبه ی نتیجه ی ضرب دو ماتریس

به صورت بازگشتی.

تعداد عملیات محاسبه: ۷ ضرب و ۱۸ 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 و کسب 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 کروسکال

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.

$F = \emptyset$; // مجموعه ی اولیه ی یال ها تهی است.

$Y = \{v_1\}$; // 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

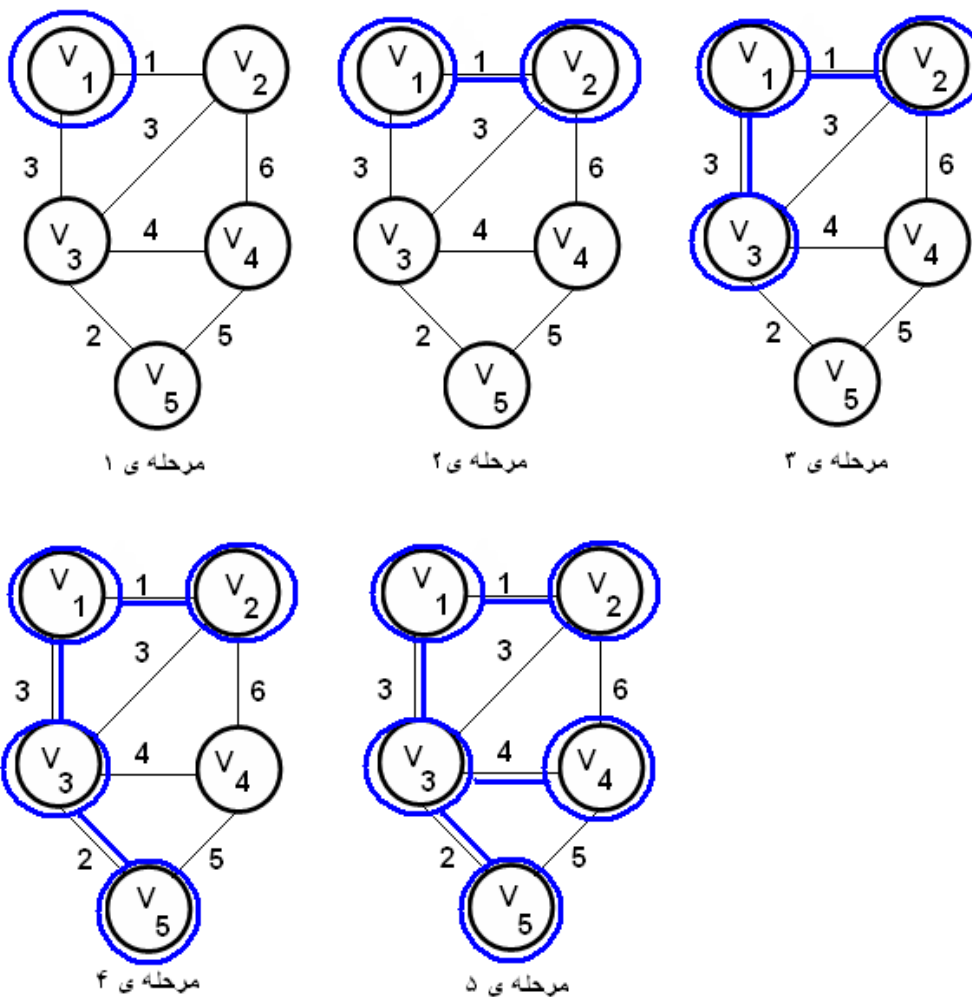
$(Y = V)$

// اگر همه ی رئوس پوشش داده شود. بررسی انتخاب

}

مسئله حل است.

مثال:



توجه: درخت پوشای مینیمال منحصر به فرد نیست. (در مرحله ۲ یال $V_2 V_3$

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.

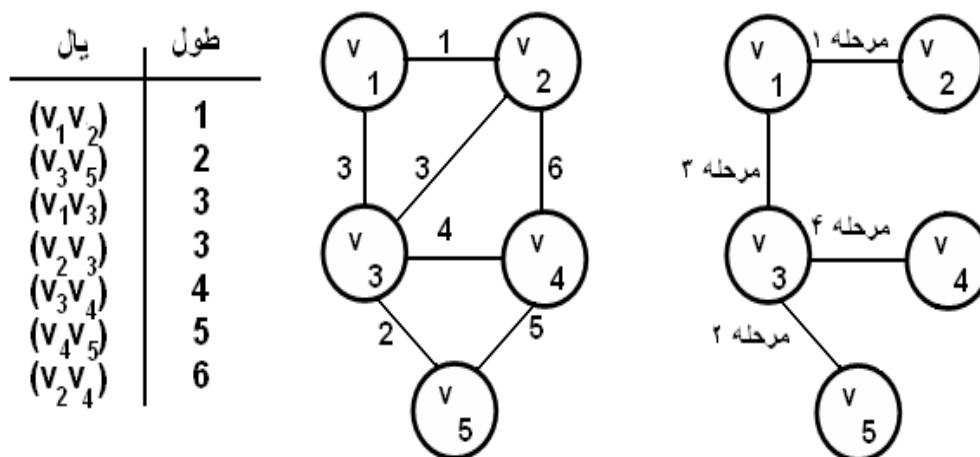
الگوریتم کروسکال :

- (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:



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 $n-1 \leq m \leq \frac{n(n-1)}{2}$

نکات تستی:

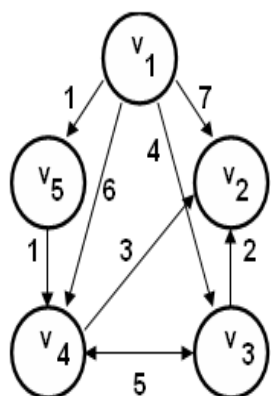
الگوریتم	پایم	کروسکال
گراف کامل	$\theta(n^2)$	$\theta(n^2 \log n)$
heap (گراف متراکم)	$\theta(n^2)$	$\theta(n \log n)$

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

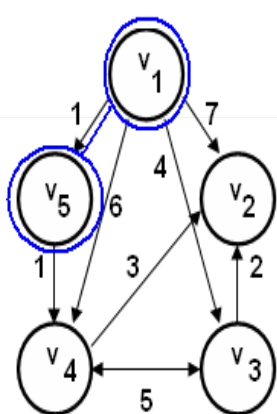
الگوریتم دیکسترا:

یافتن کوتاهترین مسیرها 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).

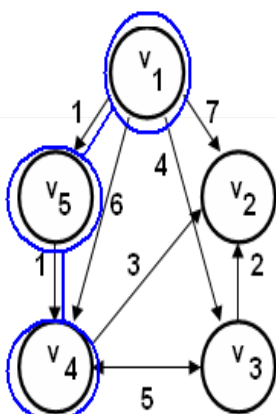
مثال: تمام مسیرها از V_1 به بقیه ی رئوس را بیابید.



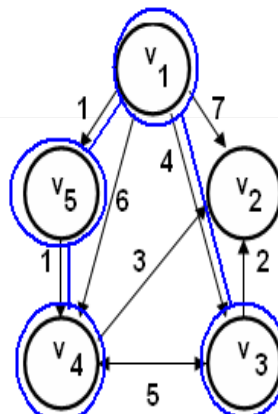
ورودی



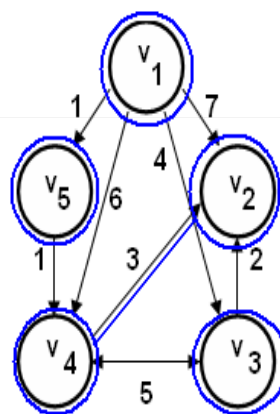
مرحله ی ۱



مرحله ی ۲



مرحله ی ۳



مرحله ی ۴

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

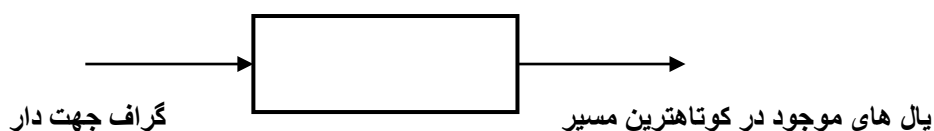
* در این روش مقداری که در هر مرحله انتخاب می کنیم باید نسبت به مبدا مقایسه گردد. علاوه بر آن چون جهت دار است باید به جهت آن نیز دقت شود.

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

ورودی: عدد صحیح $n \geq 2$ 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 از ۱ تا 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 کوتاهترین 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	۱۰ سنت
۱	۵ سنت
۱۰	۱ سنت

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) && $c \neq \emptyset$) // \rightarrow

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

{

$X = \text{select}(c)$; //

انتخاب یکی از اعضای مجموعه ورودی

$C = C - \{x\}$; //

کم کردن آن عنصر از زیر مجموعه ورودی

If (faceible (s,x))

$S = S + \{x\}$; // \Rightarrow

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



* 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 شیء i

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

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 $\sum_{i=1}^N p_i x_i$: تابع هدف
ارزش عنصر i ام (abjective)

میزان انتخاب شده از عنصر i ام، x_i می تواند صفر ، یک یا کسری بین $w_i < 1$ باشد

$\sum_{i=1}^N W_i x_i < w$: محدودیت ها (constraints)
ظرفیت کوله پشتی
وزن عنصر i ام

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 (π_i/w_i)

جواب بهینه می دهد

اگر جدول مسئله را بر حسب π_i/w_i و نزولی مرتب کنیم :

معيار انتخاب selection feceible soluction

X_i	π_i	w_i	π_i/w_i
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

X_i	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

$$\sum_{i=1}^s \pi_i x_i = 0*8 + 0*15 + 17/25*15 + 1*10/10 + 1*20/20 = 40.1$$

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

ظرفیت کوله پشتی

* شبه کد C در صفحه ۱۶۵ *

تحليل پیچیدگی الگوریتم کوله پشتی :

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

الگوریتم کوله پشتی (با استفاده از روش حریصانه :

لیست شامل سود اشیاء $P=(p_1, p_2, \dots, p_n)$

لیست شامل وزن اشیاء $W=(w_1, w_2, \dots, w_n)$

$X=(x_1, x_2, \dots, x_n)$

لیست شامل اشیاء انتخاب
نشده مقدار آنها

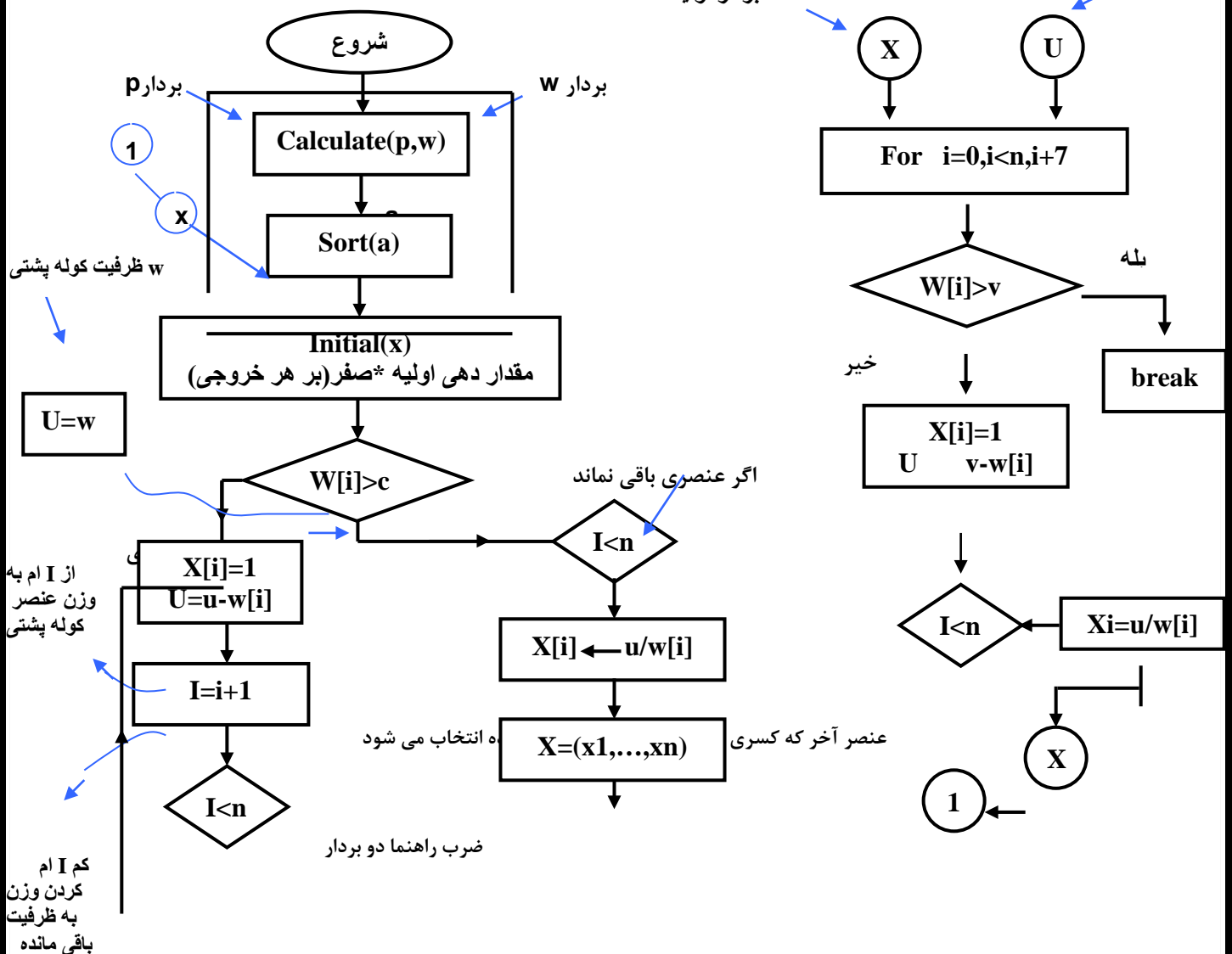
$P=(p_1, \dots, p_n)$

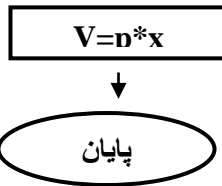
$W=(w_1, \dots, w_n)$

$A(p_1/w_1, p_2/w_2, \dots, p_n/w_n)$

مقدار اولیه وزن باقی

بردار اولیه





مسئله زمانبندی :

دو گونه زمان بندی داریم :

۱. کمینه سازی کل زمان برای انتظار کشیدن و سرویس 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

$$3! = 3 * 2 * 1 = 6$$

مثال :

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

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

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

جواب بهینه *

$$\begin{array}{l}
 \leftarrow 4+(4+5)+(4+5+10)=32 \\
 4+(4+10)+4+10+5=37
 \end{array}
 \begin{array}{l}
 (3,1,2) \\
 (3,2,1) \\
 T1=3 \quad t2=2 \quad t3=1
 \end{array}$$

بهترین حالت با کمترین زمان

* در واقع در هر مرحله ترتیب را تغییر می دهیم و برای مثال در آخری زمان انجام انجام کار 3 همان $t1$ خواهد شد زمان انجام کار دوم 2 همان $t2$ خواهد شد و زمان انجام کار 1 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 دهنده 1 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 `ascii`: تبدیل کارکترها به یک تعداد بیت ثابت مثلاً 8 بیتی در این

روش هر 8 بیت یک کارکتر در نظر گرفته می شود.

A → 00100001
B → 00100010

۲- روش هافمن : اگر برخی کارکتر ها زیادتیر 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.

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

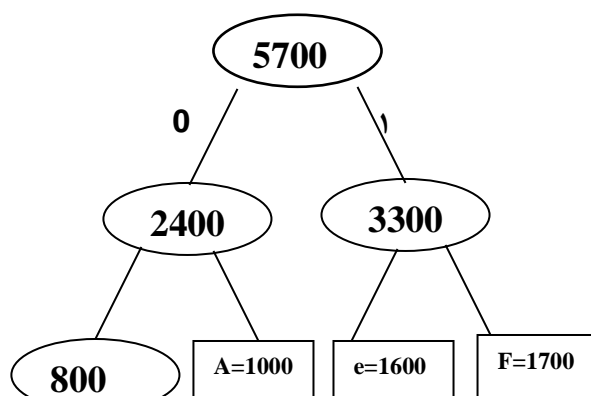
شروع :

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 houx استفاده می شود .



C=0000

D=0001

1 0 1

1

کد نوشته شده در کتاب اشتباه

است منظور همان شکل

صفحه 175

۱

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 و حل :

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

خروجی : BIN : ضریب دو جمله ای (N, K)

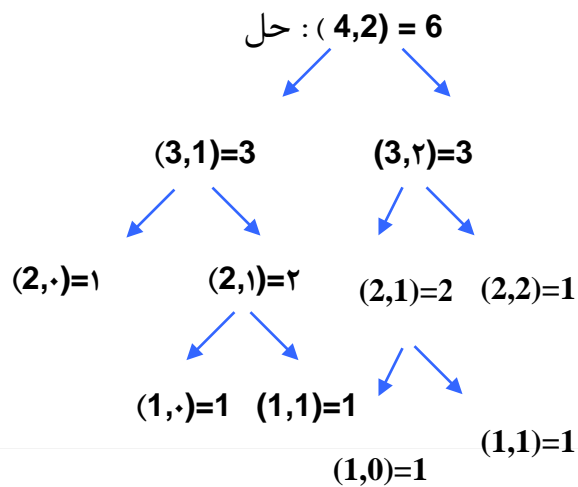
شبه کد در صفحه 203 کتاب

```
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 205 کتاب

```

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==i)
                B[i] [j] = 1 ;
            else
                B[i] [j] = B[i-1] [j-1] + B[i-1] [j] ;
    Return B[n] [k] ; }

```

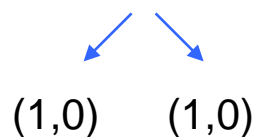
N

K

	0	1	2	3	4
0	1				
1	1	1			
2	1	2	1		
3	1	3	3	1	
4	1	4	6	4	1
5	1	5	10	6	1
6	1	6	15	10	4

N نباید از K کوچکتر باشد

(2,1)



چون در آرایه قبلا محاسبه شده

از همان ۲ استفاده می کنیم .

جواب ۲ و ۴ که در مسئله قبل از طریق

تقسیم و حل به این جواب رسیده بودیم.

جواب تکی یک پشت راه حل

متفاوت است. در مرحله قبل از

seek استفاده کردیم. پس در

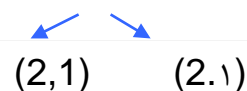
روش از craray استفاده کردیم

که بدین سبب مراحل که در از

دست داد یم و نیازی به محاسبه ی

مقادیر تکراری نیست .

(3,1)



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)

$$\begin{aligned} T(n) &= \begin{cases} (n-1, k-1) + (n-1, k) \\ k = n = 1 \end{cases} \\ t(n) &= \binom{n}{k} - 1 \\ t(n) &= \theta(nk) \end{aligned} \quad \left\{ \begin{array}{l} \text{روش تقسیم و حل} \\ \text{روش برنامه نویسی پویا} \end{array} \right.$$

مسئله : جمله n ام دنباله ی فیبوناچی

ورودی ها : $n \geq 1$

خروجی ها : جمله n ام دنباله ی فیبوناچی

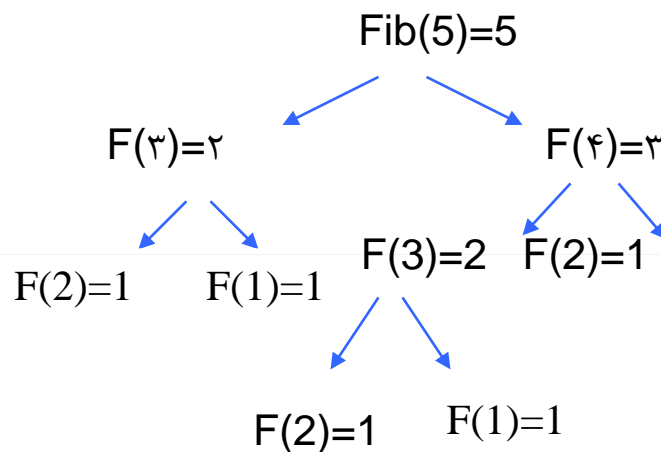
$$F(0)=0, f(1)=f(2)=1$$

$$F(n) = \text{fib}(n-1) + \text{fib}(n-2)$$

۱ ۲ ۳ ۴ ۵ ۶ ۷
۱ ۱ ۲ ۳ ۵ ۸ ۱۳

از روش تقسیم و حل : ساختار پشته

علت ناکارآمد : عملیات تکراری و محاسبات تکراری



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

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

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 4×4 بنحوی قرار دهید که هیچ ۲ وزیری یکدیگر را تهدید نکنند .

حل : روش متعارف و عادی

شرط تهدید نکردن ۲ وزیر :
 ۲.۱ وزیر در یک سطر نشینند
 ۲.۲ وزیر در یک ستون نشینند.
 ۲.۳ وزیر در یک سطر نشینند.

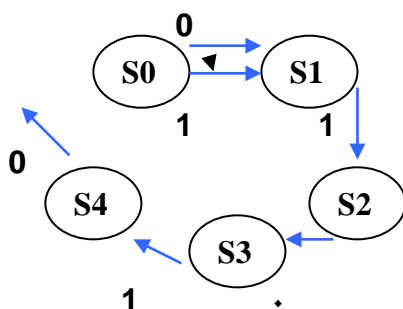
$$16 \cdot 15 \cdot 14 \cdot 13 = 16! / 12!$$

تعداد کل حالات

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

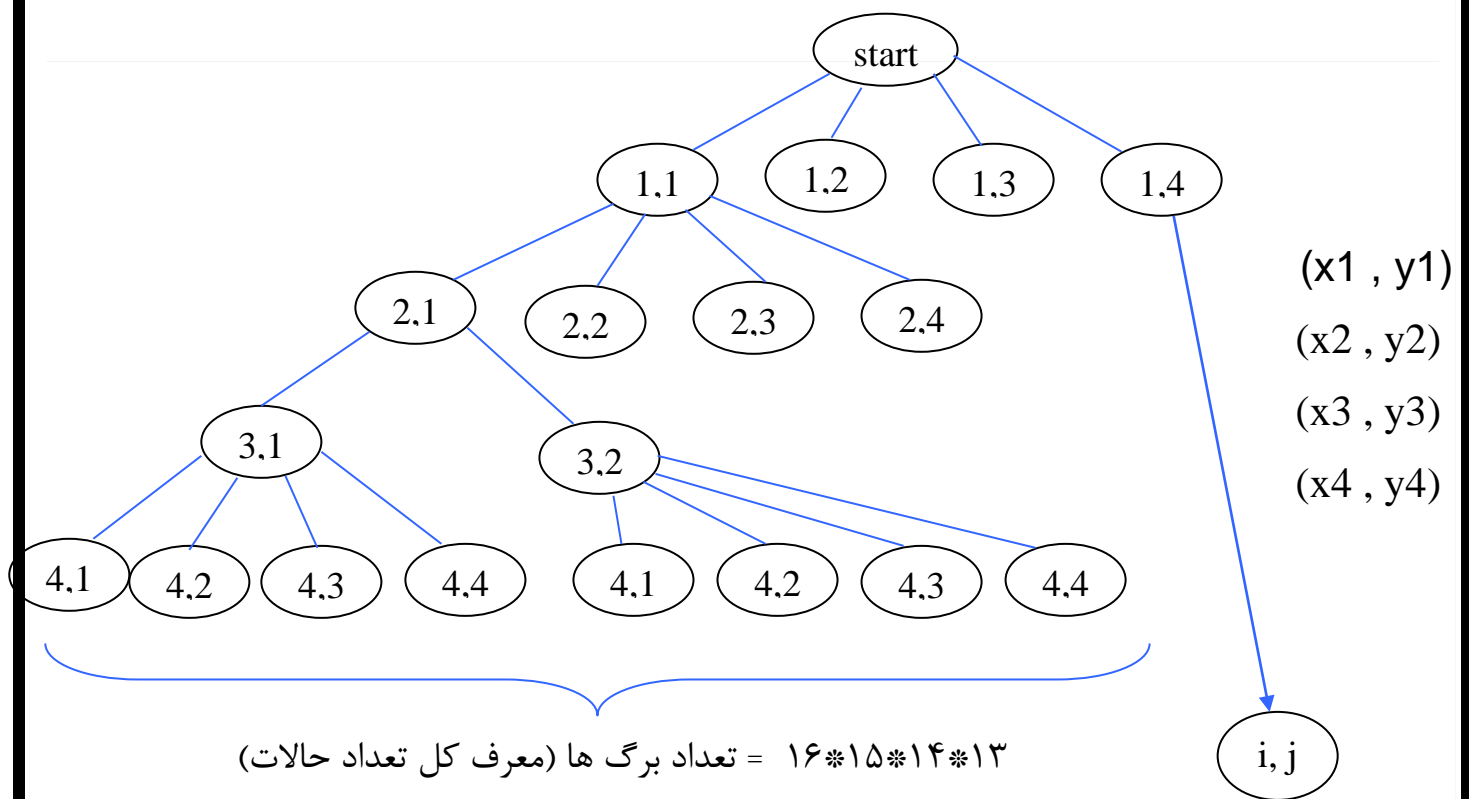
تعریف حالت (state):

اگر متغیرهای مسئله x_1, \dots, x_i باشند رابطه (x_1, \dots, x_n) معرف حالت است . حالت مسئله در حقیقت وضعیت فعلی آن سیستم را نشان می دهد ، از مفاهیم مهم در نظریه سیستم 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 .



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

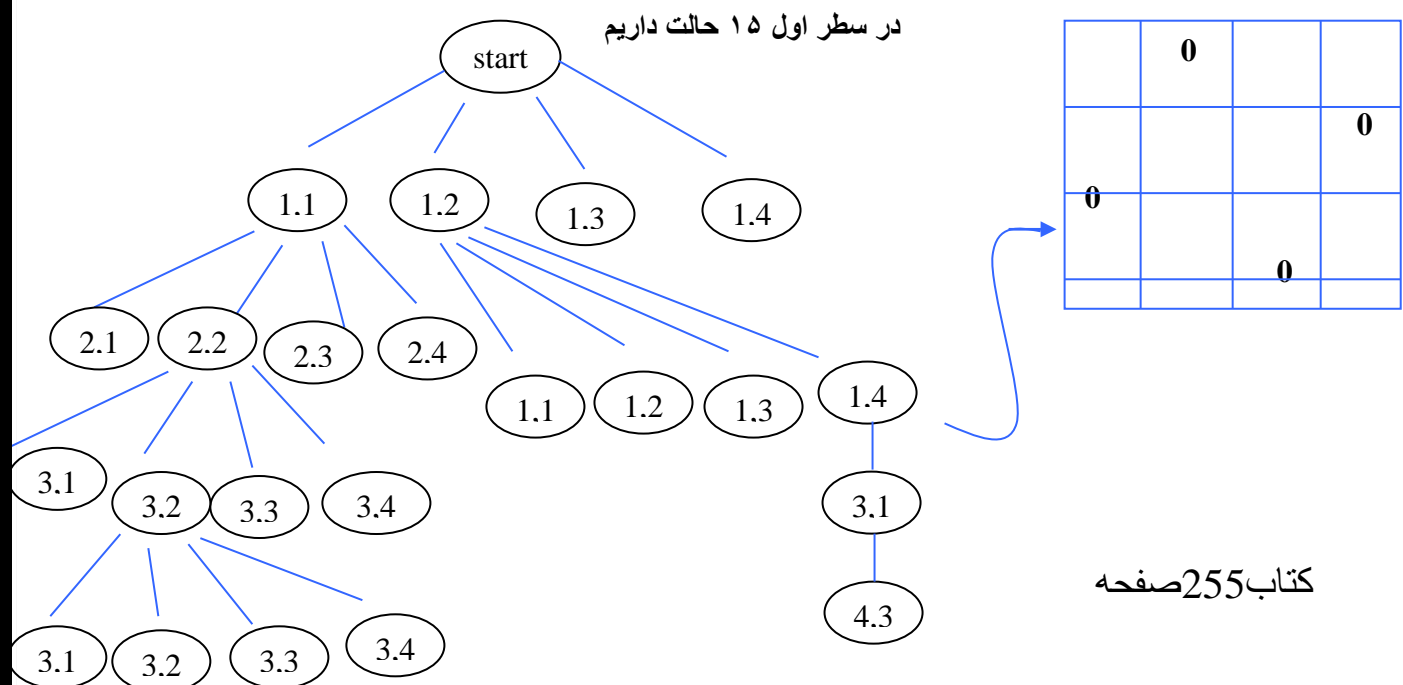
(Game theory) نظریه زبان ها ← ytratoyy

MCDM : multiple cvriteria 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
 درخت حالت با تعریف متغیرهای جدید :
 1,1,2,2) شماره ستون با این پیش فرض که شماره سطر یکی نیستند
 (1,1) (1,2) (2,1) (2,3)



تعداد کل حالت ها (برگ ها)

$$4 * 4 * 4 * 4 = 4^4$$

تعداد حالات قرار گرفتن وزیر ۱ در سطر ۱

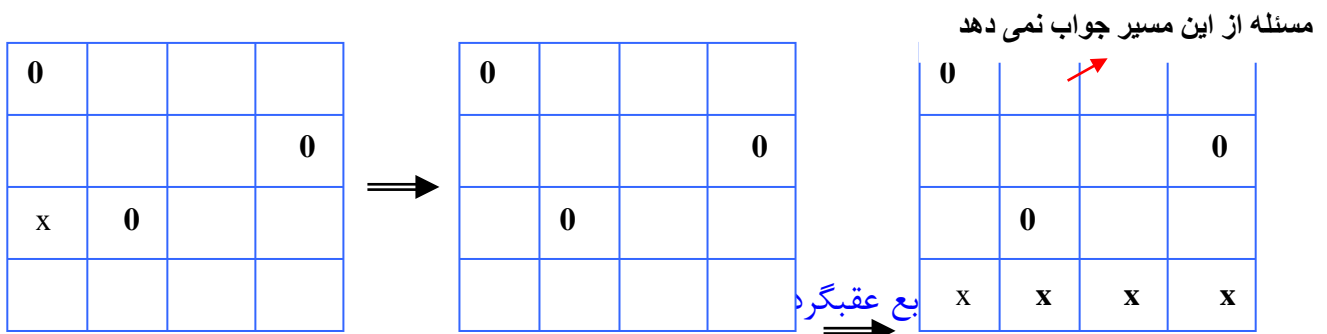
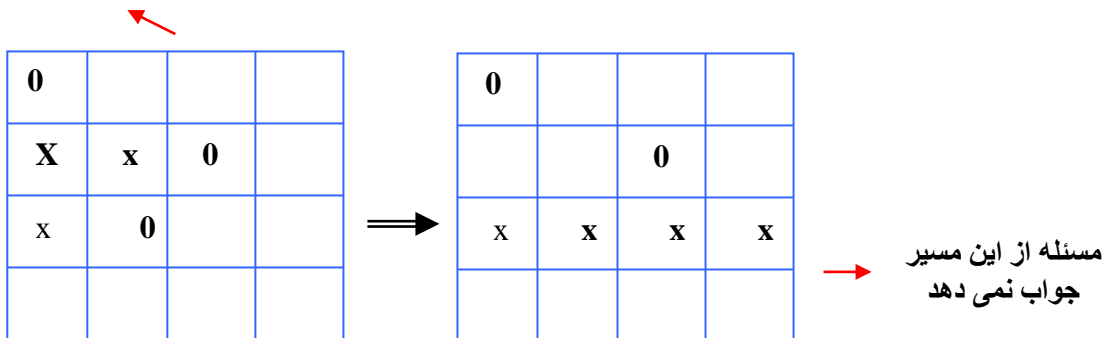
تعداد حالات قرار گرفتن وزیر ۲ در سطر ۲

تعداد حالات قرار گرفتن وزیر ۳ در سطر ۳

تعداد حالات قرار گرفتن وزیر ۴ در سطر ۴

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

امید بخش نیست لذا پایین تر نمی رویم



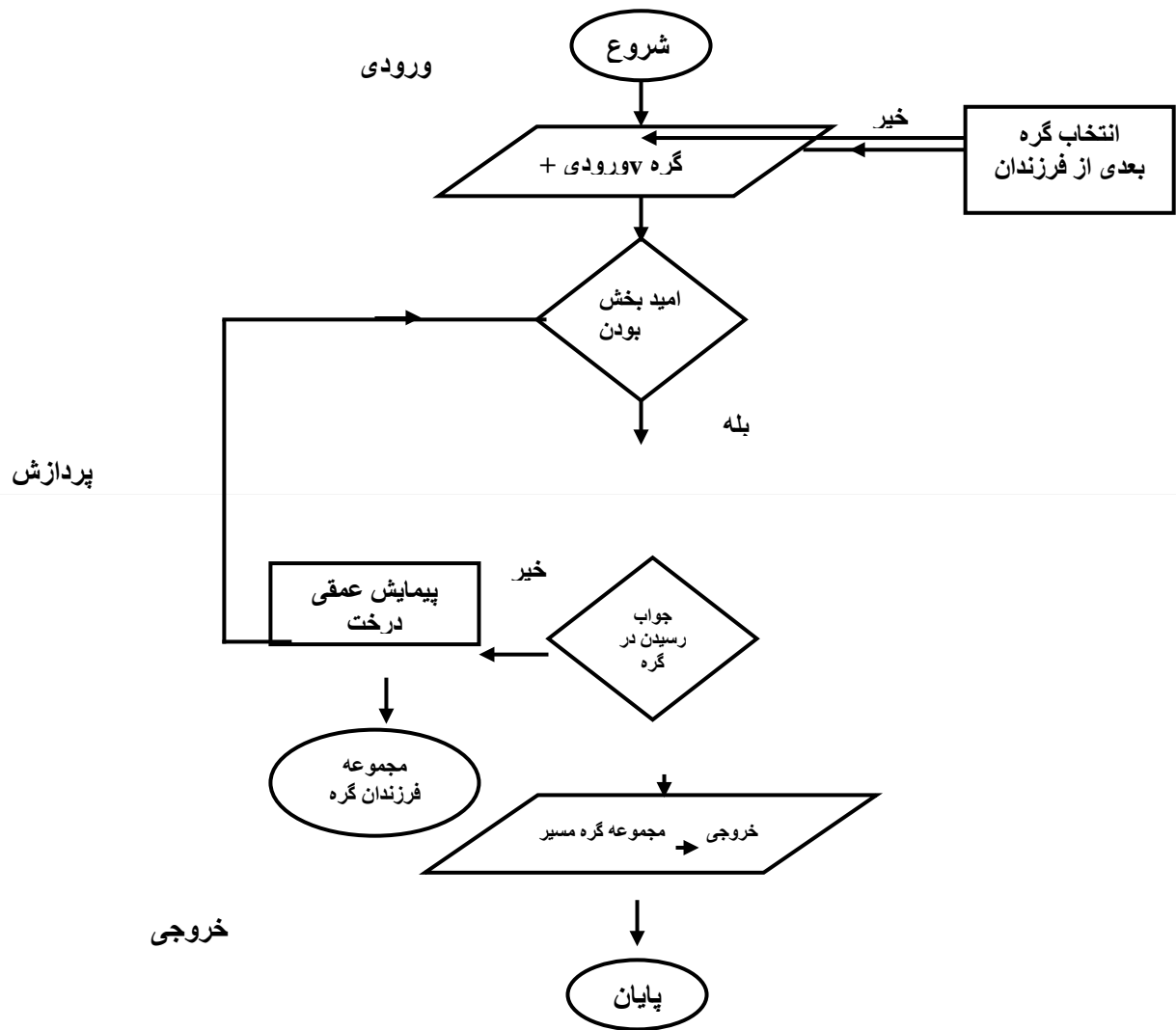
ورودی : درخت حالت

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

ورودی
شبه کد در کتاب
pseudocode



خروجی
یک مسیر از ریشه برگ



```

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) (x_1, x_2, \dots, x_n) یک جواب از یک جواب از فضای حالت (شماره سطر وزیر معین و شماره ستون وزیر معین)

$j=1$	$col(i)=col(k)$ یا $X_i=X_k$	1-بررسی ستونها
شماره ستون ۲ وزیر یکی است		
$i-j=k-l$ $i+j=k+l$ $i-k=k-j$	$col(i)-col(k)=i-k^*$ یا $x_i-x_k=i-k$ $Col(i)+col(k)=k+i^*$ یا $X_i+X_k=k+i$	۲-بررسی سطرها
۲ وزیر در سطر همدیگر را تهدید می کنند		

یک جواب از فضای حالت $(x_1, x_2, x_3, x_4)=(2, 4, 1, 3)$ فرمول صفحه ۲۵۷ کمی مبهم است اصلاح شود .

		0	
		0	0
0			0
		0	

صورت مسئله : تا این مرحله مقادیر x_1 تا $k-1$ مشخص شده اند می خواهیم ببینیم که آیا مقدار k امیدبخش است یا نه

مقادیر قبلی

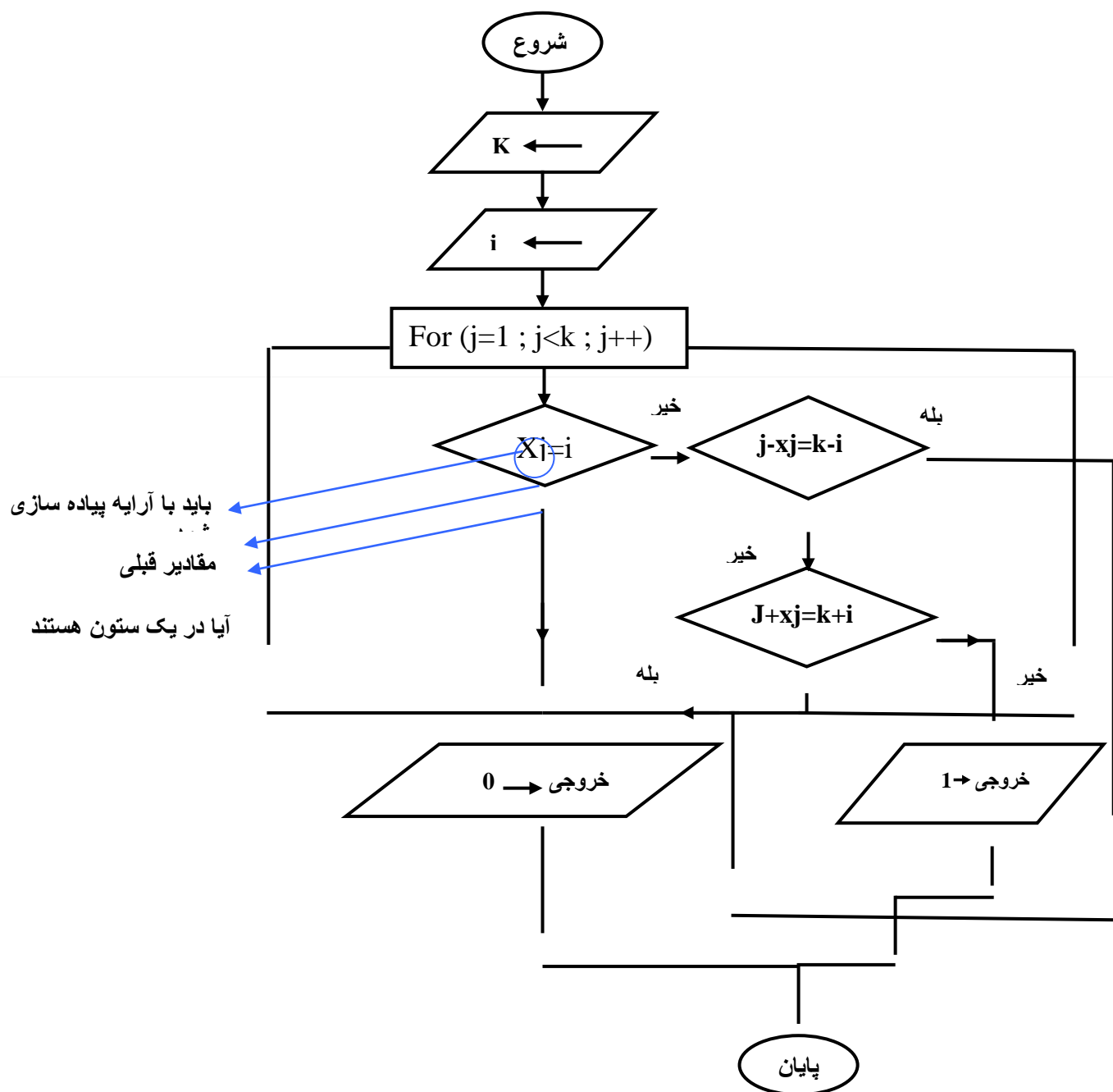
$$(x_1, x_2, \dots, x_j, \dots, k, \dots)$$

\swarrow j \swarrow شماره ستون فعلی

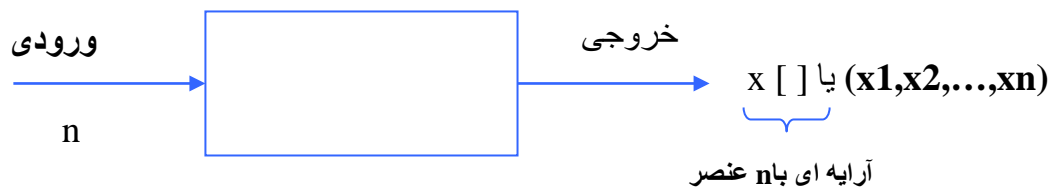
توجه شود در صفحه قبل { شماره سطر اول j شماره ستون اول است. K شماره سطر دوم k شماره ستون دوم }

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

شماره سطر شماره ستون

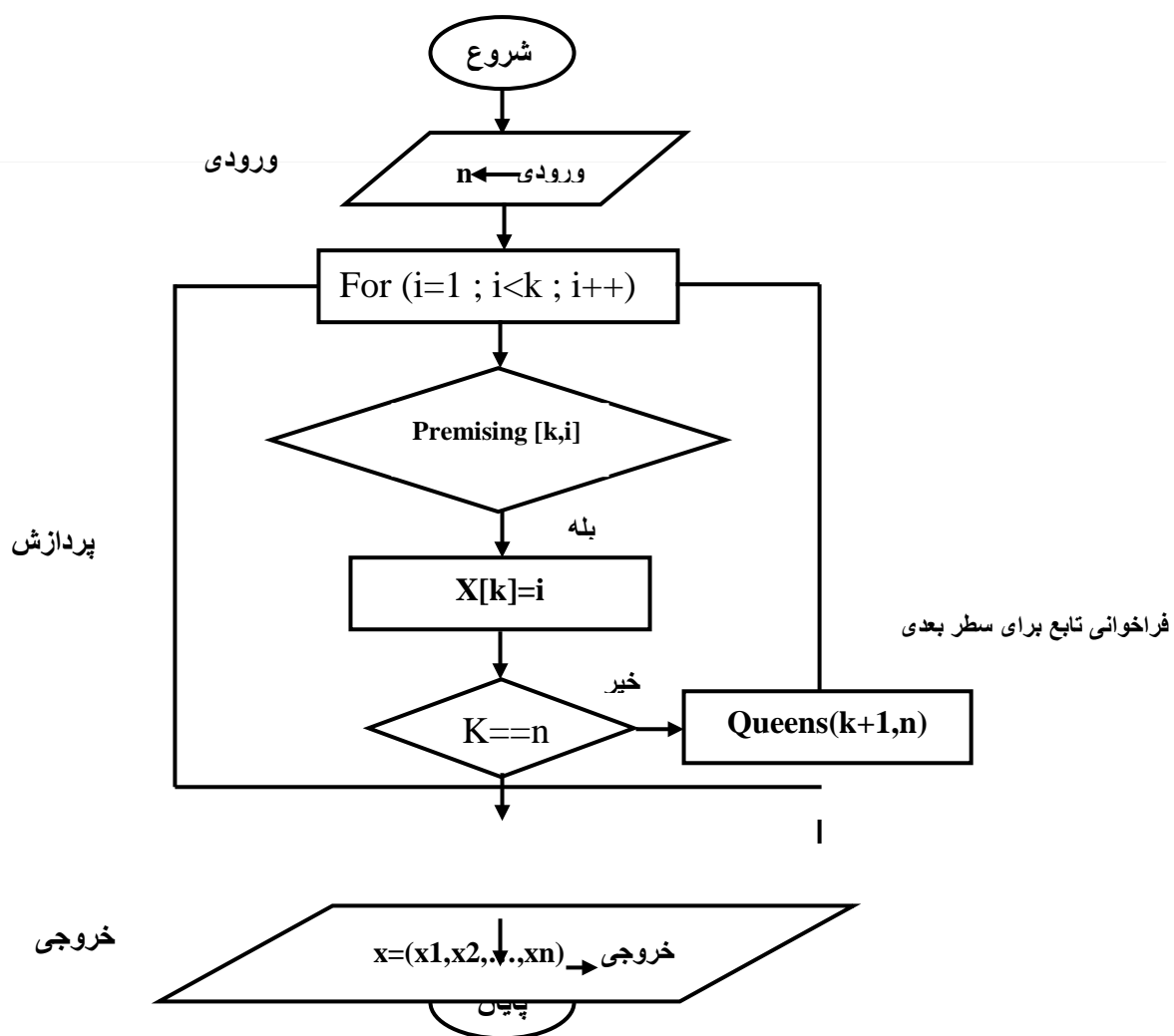


الگوریتم عقبگرد برای مسئله n وزیر:



Queens (k , n)

تعداد وزیرها شماره سطر و کد 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 نماید .

مسئله حاصل جمع زیرمجموعه ها :

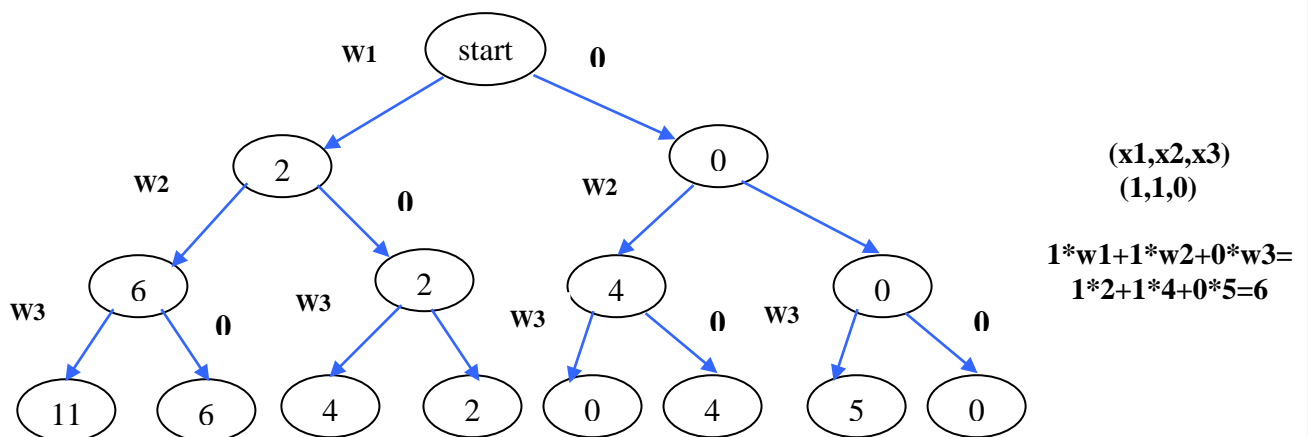
صورت مسئله: ورودی: n عدد صحیح مثبت مقادیر $w_1, w_2, w_3, \dots, w_n$
خروجی: (x_1, x_2, \dots, x_n) یا x_i 0 یا 1 است

مثال :
 $\begin{cases} w_1=5, w_2=6, w_3=15, w_4=11, w_5=16 \\ W=21 \end{cases}$

$$\begin{cases} W_1 + w_2 + w_3 = 21 \\ W_1 + w_2 = 21 \\ W_3 + w_4 = 21 \end{cases}$$

مجموعه جواب $\{W_1, w_2, w_3\} \quad \{w_1, w_2\} \quad \{w_3, w_4\}$

مثال دیگر:
 $\begin{cases} w=6 \\ w_1=2, w_2=4, w_3=5 \end{cases}$
 بهتر است ابتدا بصورت صعودی بچینیم



بررسی امید بخش بودن یک گره:

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

اضافه کردن گره پایینی از مرز مجاور

$$weight + w_{i+1} > w \quad .1$$

$$weight + total < w \quad .2$$

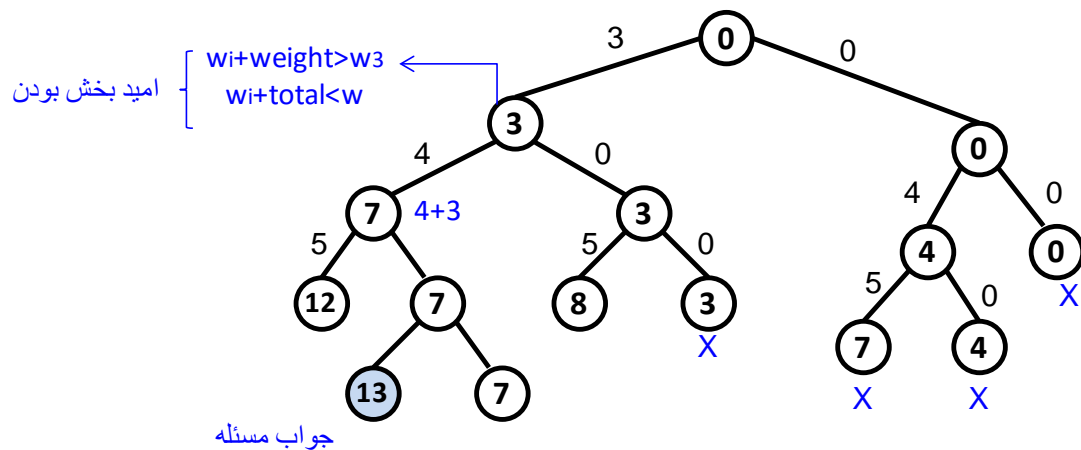
بررسی گره در سطح A

هر چقدر پایین تر برویم به وزن مطلوب
نهایی نمی رسیم

ردیف	پارامتر	
۱	w	مجموع وزن نهایی
۲	$weight$	مجموع 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

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

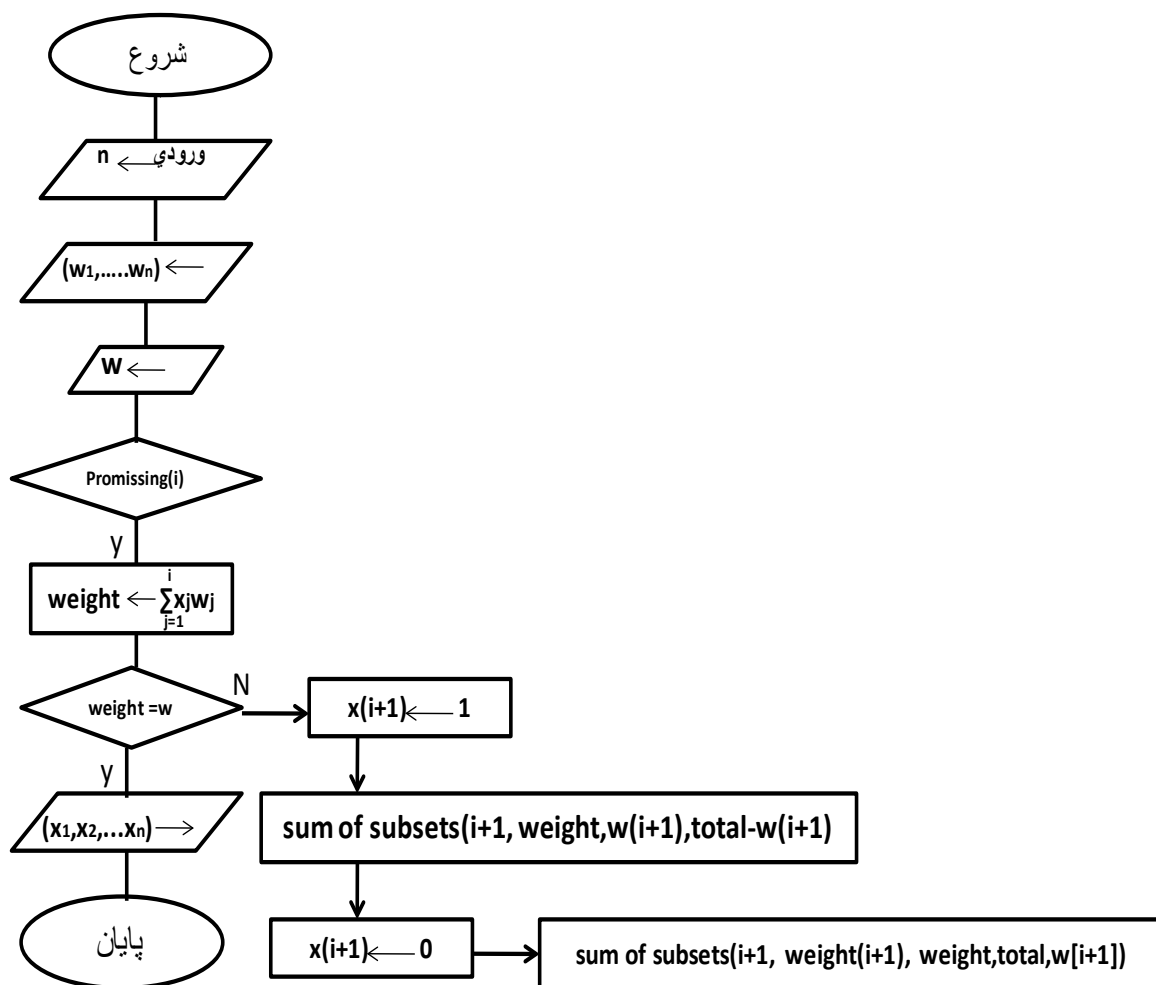


* موارد X غیر امیدبخش هستند.

sum of subsets(i, w, total)

الگوریتم مسئله جمع زیر مجموعه ها

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



الگوریتم امید بخش بودن:

```
bool promising(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) →

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

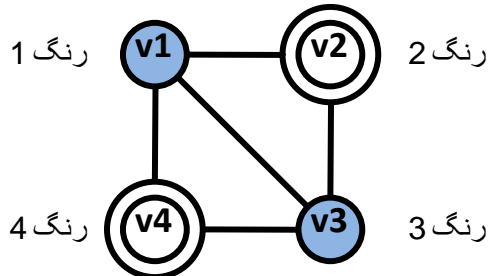
اولین فراخوانی تعداد گره های هرس شده

$$1+a+a^2 + \dots + a^n = \frac{a^{n+1} - 1}{a - 1}$$

فرمول جمع سری هندسی یا توافقی

رنگ آمیزی گراف:

مسئله: 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 مجاور هم رنگ نباشند

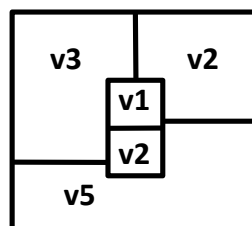
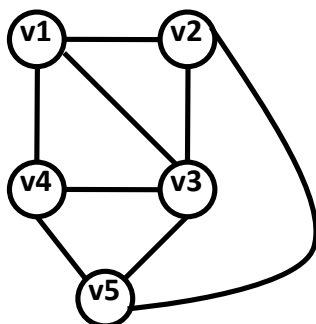


گرافی که با دو رنگ قابل رنگ آمیزی نیست

(حل مسئله با سه رنگ)

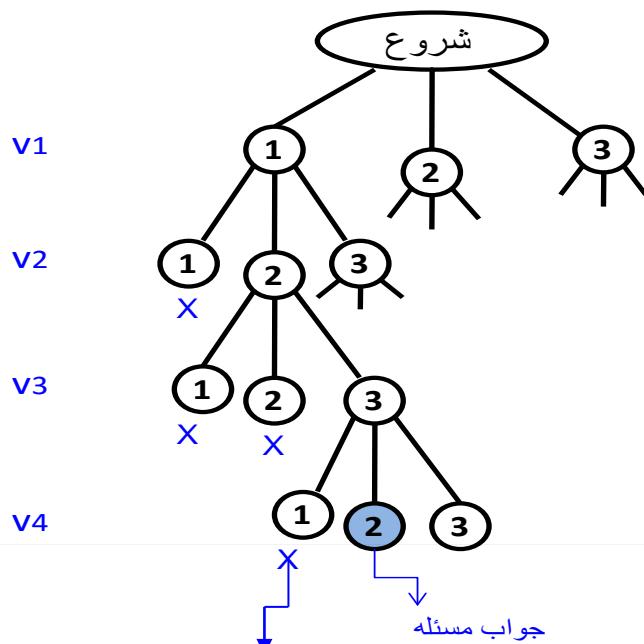
کاربرد رنگ آمیزی نقشه:

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

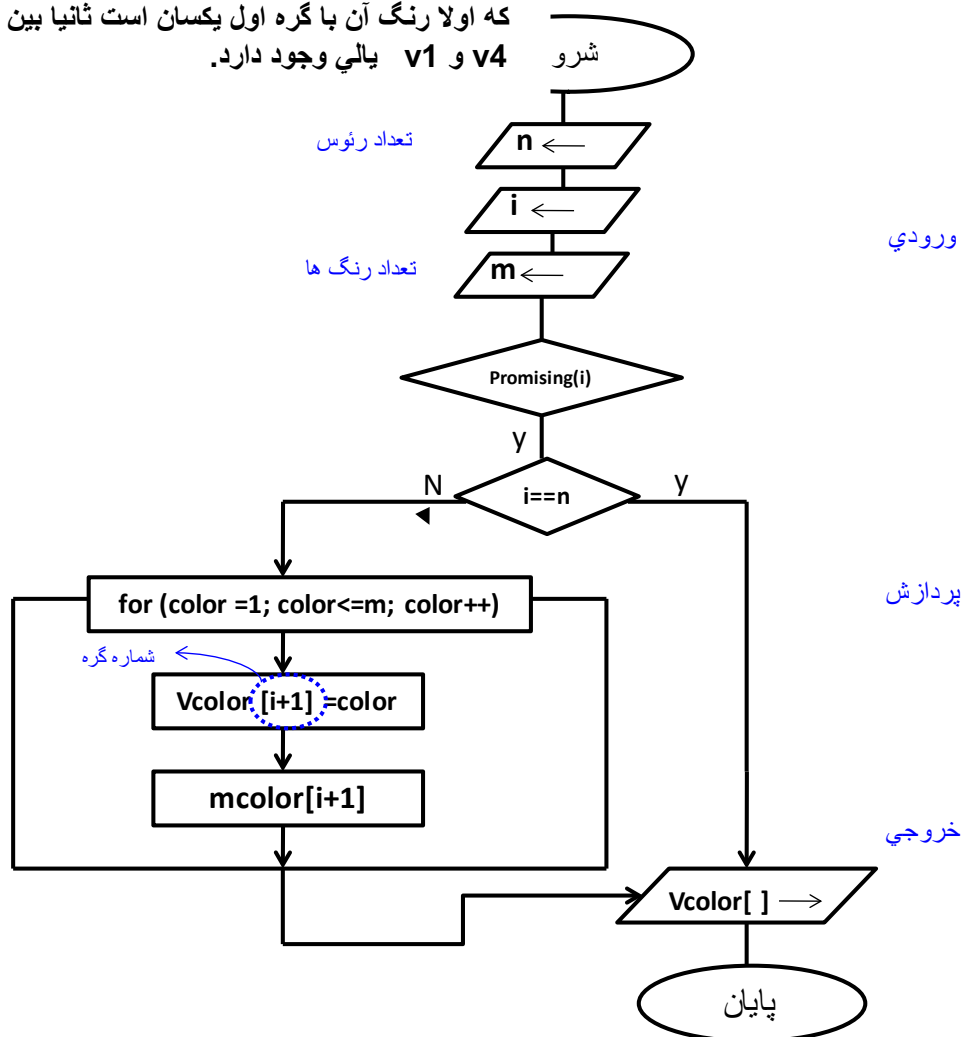


یک نقشه وگراف مسطح هم ارز آن

رنگ آمیزی گراف با سه رنگ:



به طور مثال دلیل این که این گره غیر امیدبخش است این است که اولاً رنگ آن با گره اول یکسان است ثانیاً بین دو راس $v1$ و $v4$ یالی 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 یکی باشد

زوا
بین دو
راس
ارتباط
باشد

پیچیدگی زمانی:

vcolor(0)

فراخوانی تابع در بالاترین سطح

تعداد گره ها در درخت فضای حالت:

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

مسئله دور هامیلتونی:

مسیری که از هر راس فقط یک بار عبور می کند و به راس شروع برمی گردد

یادآوری: مسئله تعیین کوتاهترین تور با n=20

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

$(n-1)!$

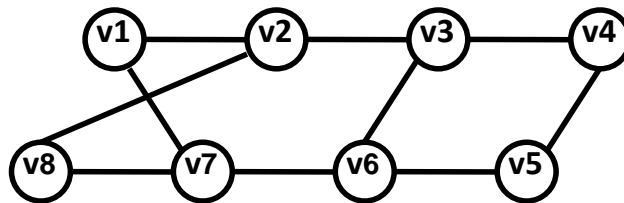
- روش کورکورانه:

6.46 سال	روش برنامه ریزی پویا	n=40
----------	----------------------	------

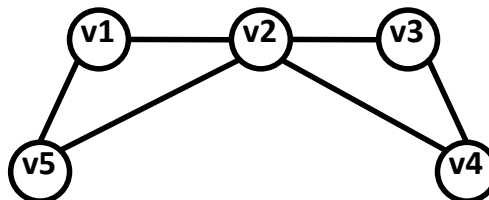
*مسئله یافتن هر توری در گراف (بدون توجه به وزن تور) را مسئله دورهای

هامیلتونی می نامند

الف) دارای دور هامیلتونی



ب) دارای دور هامیلتونی



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

مسئله کوله پشتی صفر ویک:

- گره غیر امید بخش $weight \geq w$

- مرتب سازی قطعات بر حسب pi/i به صورت غیر نزولی

- تعیین حد بالا برای سود قابل حصول از گسترش دادن گره (bound)

- profit: حاصل جمع ارزش قطعاتی که تا آن گره در نظر گرفته شده اند

- weight: حاصل جمع اوزن آن قطعات

- مقدار اولیه متغیر bound را برابر profit قرار می دهیم.

- مقدار اولیه متغیر total weight را برابر weight قرار می دهیم.

- هر بار به روش حریصانه یک قطعه را برداشته و ارزش 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 از w 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 k ام باعث می شود جمع از حد مجاز بیشتر شود

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

بهره حاصل از $k-1$ قطعه نخست (که مطمئنیم مجموع وزنهای k بیشتر نیست)
ظرفیت باقی مانده برای قطعه لازم
بهره واحد وزن برای قطعه k ام

* گره غیر امید بخش: $\text{bound} \leq \text{maxprofit}$ (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 \$

$\text{max profit} = 0$

$\text{profit} = 0$

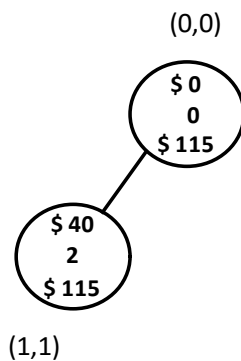
$\text{weight} = 0$

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

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

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

* گره امید بخش است چون وزنش کمتر از $w=16$ و حدش بزرگتر از maxprofit است.



$$\text{Profit} = \$0 + \$40 = \$40 > \text{maxprofit} = 0 \rightarrow \text{maxprofit} = \$40$$

$$\text{weight} = 0 + 2 = 2$$

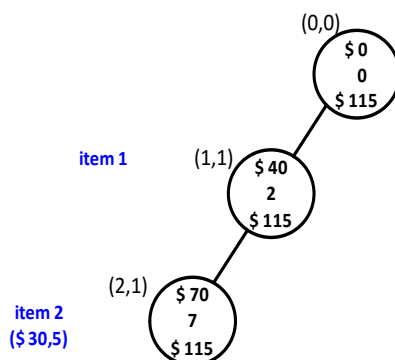
$$\text{totalweight} = \text{weight} + \sum_{j=i+1}^{3-1} w_j = 2 + 5 = 7$$

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

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

* گره امید بخش است چون وزنش کمتر از $w=16$ و حدش بزرگتر از maxprofit است.

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



$$\text{Profit} = \$40 + \$30 = \$70 > \text{maxprofit} \rightarrow \text{maxprofit} = \$70$$

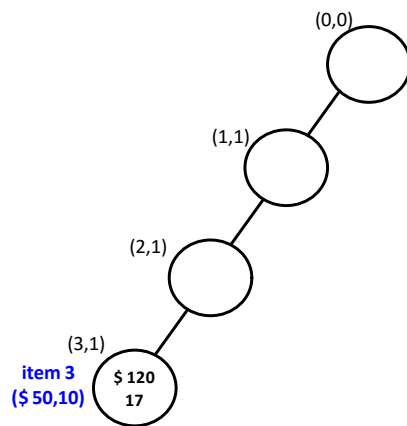
$$\text{weight} = 2 + 5 = 7$$

$$\text{maxprofit} = \$70$$

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

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

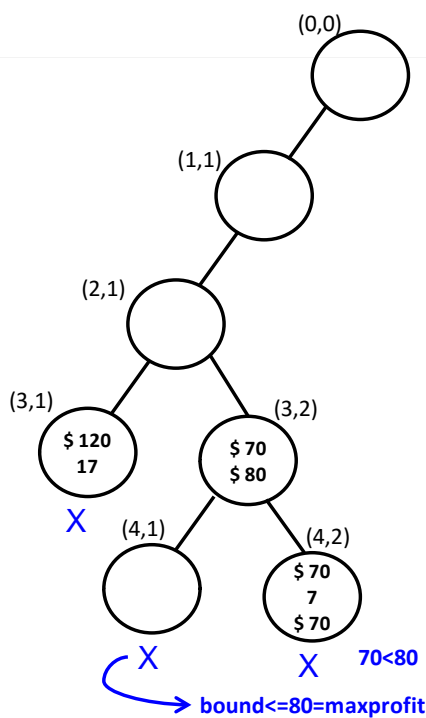
* گره امید بخش است چون وزنش کمتر از $w=16$ و حدش بزرگتر از maxprofit=70 است.



Profit = \$70 + \$50 = \$120
 weight = 7 + 10 = 17 > 16
 maxprofit = \$70

غير امید بخش
 تغییر نمی کند

* چون وزن بیشتر از حد مجاز است نیازی به محاسبه bound نیست



Profit = \$70
 weight = 7
 maxprofit = \$70

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

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 مسیر از 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:

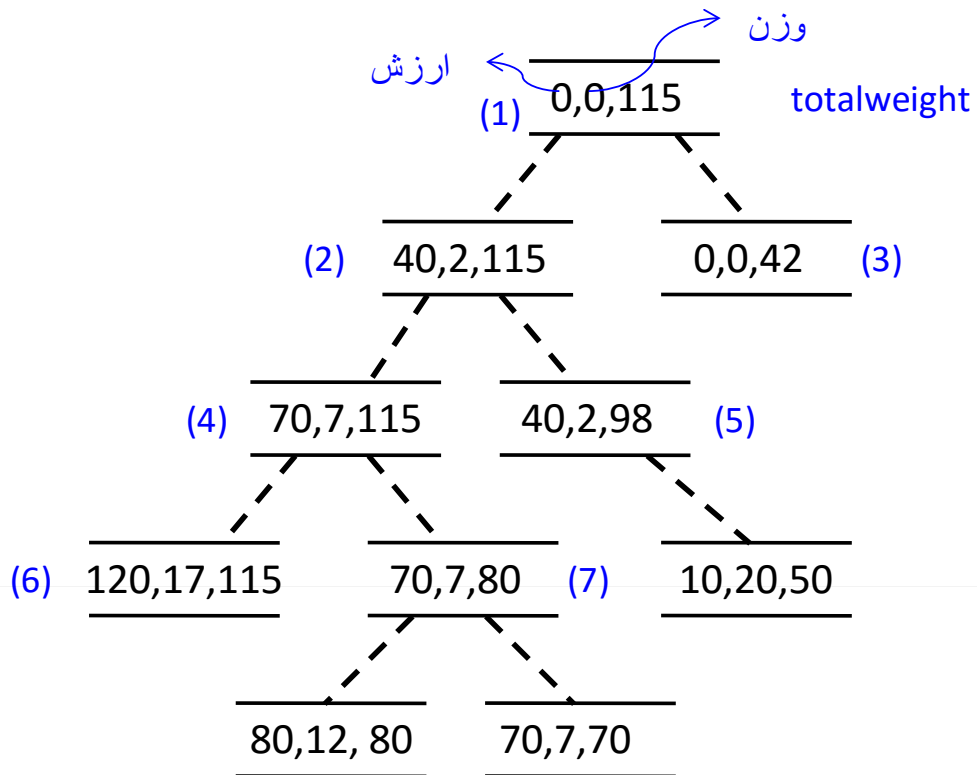
۱. ما را به روش خاصی از پیمایش درختی محدود نمی کند
۲. تنها برای مسائل بهینه سازی به کار می رود
۳. جستجو بر اساس اول و بهترین انجام می پذیرد.

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	p_i	w_i	p_i / w_i
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 solver در Excel: در بخش solver, 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

* سیستم Decision support system: Dss : سیستم تصمیم سازی 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 و خروج هر گره در آن تور و تقسیم حاصل جمع بر ۲

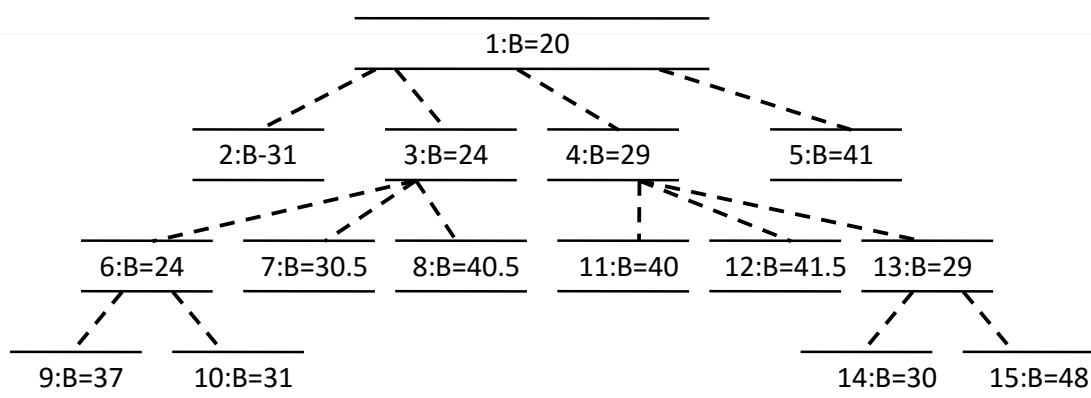
روش سطرو ستونی:

	1	2	3	4	5
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

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

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

* روش دیگر برای محاسبه B: $14 + 7 + 2 + 4 + 2 + 4 = 31$



محاسبه هزینه تور:

۱. تا لبه (1,2) به صورت قطعی انتخاب شده است از روش ورودی به گره ۳ و خروجی 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 Np	مثال از p
Sum of subset	۱. تشخیص مرتب بودن یک آرایه ۲. تشخیص این که یک رشته مفروض شامل زیر رشته خاص هست یا نه

نکته: هر مسئله p متعلق به Np نیز می باشد به عبارتی $p \leq Np$ زیرا اگر برای 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 $2, n! \dots\dots$ (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 و ثابت شده که الگوریتم چند جمله ای برایشان

وجود ندارد.

* اکثر مسائل **Np complete** هستند.

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

مسئله زمان بندی:

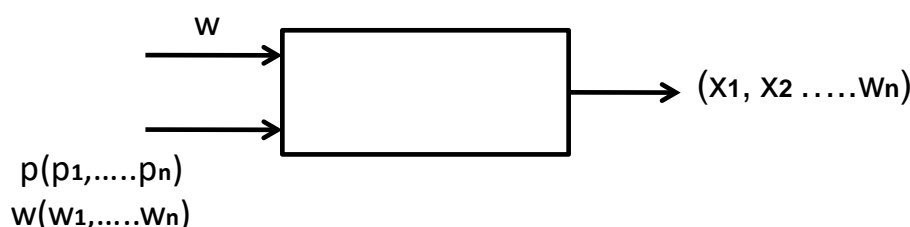
دو گونه زمان بندی داریم:

۱. کمینه سازی کل زمان برای انتظار کشیدن و سرویس دهی کارها

زمان بودن در سیستم **time in the system**

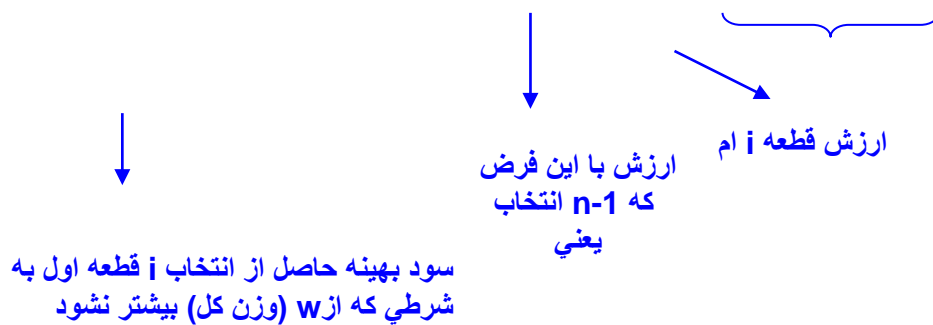
مثال: آرایشگاه (اصلاح سر)

کوله پشتی با برنامه نویسی پویا:



ارزش قطعات از اول تا $i-1$

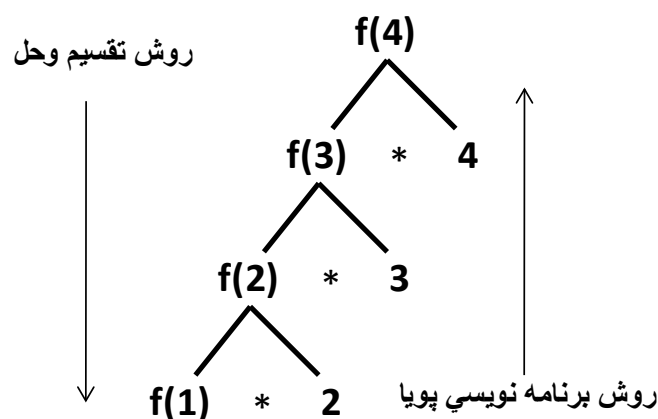
$$p[i][w] = \begin{cases} \text{maximum} (p[i-1][w], p_i + p[i-1][w - w_i]) & w_i \leq w \\ p[i-1][w] & w_i > w \end{cases}$$



* اگر w_i (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 و در غیر این صورت قطعه مورد نظر اضافه نمی شود.

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



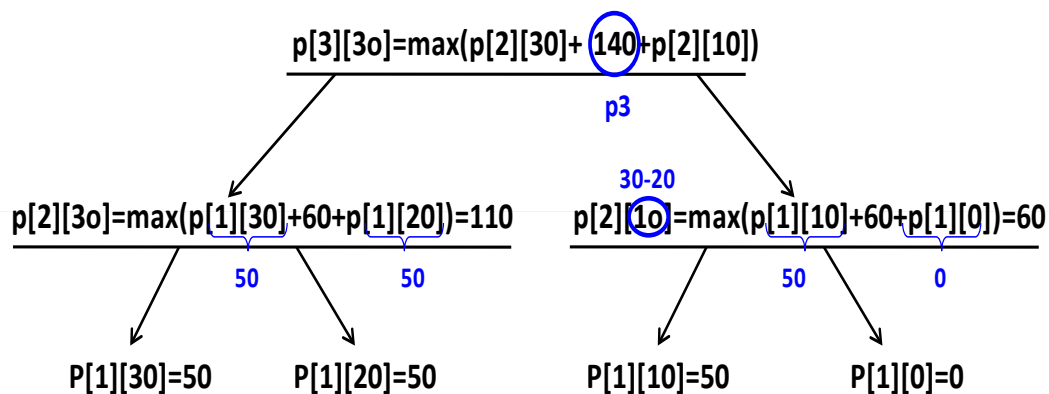
برنامه نویسی پویا

$$\begin{cases} F[1]=1 \\ \text{for } (i=1; i \leq 10; i++) \\ f[i]=i * f[i-1] \end{cases}$$

روش تقسیم و حل

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

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



* 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

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

تابع

$g_i(y)$: 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:

$$\left. \begin{array}{l} g_i(y) = 0 \text{ (از جسم } n+1 \text{ به بعد جسمی وجود ندارد که سودی داشته باشد)} \\ y < 0 \text{ (گنجایش کوله پشتی منفی باشد یعنی بیشتر از حد کوله پشتی جسم داریم)} \\ \text{و در این صورت } g_i(y) \text{ حالت ضرر دارد} \end{array} \right\} g_i(y) = -\infty$$