

به نام آن که جان را حکمت آموخت

رسانی
دینک



داده‌ساختارها و الگوریتم‌ها

Data Structures and Algorithms

محمد قدسی

دانشکده‌ی مهندسی کامپیوتر

دانشگاه صنعتی شریف

فهرست

۱	روش‌های تحلیل الگوریتم‌ها
۱	۱. زمان اجرای الگوریتم‌ها
۲	۲. بهترین و بدترین حالت و حالت متوسط
۴	۳. مرتبه‌ی الگوریتم‌ها
۵	۴. تابع‌های رشد
۹	۵. روش‌های تحلیل الگوریتم‌ها
۹	۵.۱. الگوریتم‌های ترتیبی
۱۰	۶. الگوریتم‌های بازگشتی
۱۱	۶.۱. ۱. مسئله‌ی برج‌های هانوی
۱۱	۷. روش‌های حل رابطه‌های بازگشتی
۱۱	۷.۱. ۱. حدس و استقرا
۱۶	۷.۲. تکرار با جای‌گذاری
۱۸	۷.۳. قضیه‌ی اصلی
۱۹	۷.۴. رابطه‌های بازگشتی همگن
۲۲	۷.۵. رابطه‌ها بازگشتی غیر همگن با ضرایب ثابت:
۲۲	۷.۶. حل رابطه‌ها بازگشتی غیر همگن:
۲۷	۸. مرتب‌سازی و مرتبه‌ی آماری
۲۷	۸.۱. الگوریتم‌های مرتب‌ساز
۲۸	۸.۲. درخت تصمیم

یک

(3)

۲۱	الگوریتم های مرتب ساز خطی	۲.۲
۲۱	Count Sort	۱.۲.۲
۲۲	الگوریتم Radix Sort	۲.۳.۲
۲۳	الگوریتم Bucket Sort	۲.۲.۲
۲۴	الگوریتم های مرتب ساز مبتنی بر مقایسه	۳.۲
۲۴	quicksort	۱.۳.۲
۲۷	گونه های تصادفی quicksort	۲.۳.۲
۴۰	الگوریتم HeapSort	۴.۲
۴۰	ویزگی های پک heap	۱.۴.۲
۴۴	میانه ها و مرتبه های آماری	۵.۲
۴۵	کمینه و بیشینه	۱.۵.۲
۴۶	انتخاب در زمان متوسط خطی	۲.۵.۲
۴۸	انتخاب در بدترین حالت زمانی خطی	۳.۵.۲
۵۱	مرتب سازی خارجی	۶.۲
۵۲	مرتب سازی خارجی مبتنی بر ادغام	۱.۶.۲
۵۴	مرتب سازی خارجی Polypphase	۲.۶.۲
۵۷	داده ساختارها	۳
۵۷	دسته بندی	۱.۳
۵۸	داده ساختارها برای فرهنگ داده ای	۲.۳
۵۸	درخت دودویی جستجو	۱.۲.۳
۶۱	متوسط ارتفاع درخت دودویی جستجو	۲.۲.۳
۶۳	صف اولویت (Priority Queue)	۳.۲
۶۵	تبدیل الگوریتم های بازگشتی به غیر بازگشتی	۴.۲
۶۸	برنامه ای غیر بازگشتی	۵.۲
۷۰	حذف آخرین بازگشت (Tail Recursion)	۱.۵.۳
۷۲	درختهای قرمز - سیاه (Red-Black Tree)	۶.۲

۷۲	خواص درخت قرمز - سیاه	۱.۶.۳
۷۳	دوران	۲.۶.۳
۷۴	درج	۳.۶.۳
۷۵	حذف	۴.۶.۳
۸۳	روش‌های طراحی الگوریتم‌ها	۴
۸۴	استقرای ریاضی	۱.۴
۹۰	خطاهای معمول در اثبات با استقرای متناوب	۱.۱.۳
۹۲	طراحی الگوریتم با استقرای کدگری	۲.۴
۹۷	رابطه‌ی مستقل از حلقة برای اثبات درستی الگوریتم	۱.۲.۴
۱۰۰	محاسبه‌ی مقدار یک چندجمله‌ای	۲.۲.۴
۱۰۲	زیرگراف القابی پیشینه	۳.۲.۴
۱۰۲	نگاشت یکمیمه‌یک	۴.۲.۴
۱۰۳	مسئله‌ی «ستاره‌ی مشهور»	۵.۲.۴
۱۰۴	روش تقسیم‌وحل برای طراحی الگوریتم‌ها	۶.۲.۴
۱۰۶	فرش‌گردن صفحه‌ی شطرنجی	۱.۳.۴
۱۰۶	زمان‌بندی دوره‌ی بازی‌ها	۲.۳.۴
۱۰۶	مسئله‌ی برج‌ها	۳.۳.۴
۱۱۰	ضرب در چندجمله‌ای	۴.۳.۴
۱۱۲	شبکه‌های مرتب‌ساز	۵.۳.۴
۱۱۳	روش برنامه‌ریزی پویا	۴
۱۲۱	ترکیب m از n	۱.۴.۴
۱۲۱	ضرب ماتریس‌ها	۲.۴.۴
۱۲۴	مثلث بندی بهینه‌ی یک چندضلعی محدبه	۳.۴.۴
۱۲۷	برگزگترین زیردنباله‌ی مشترک	۴.۴.۴
۱۳۱	درخت دودویی جست‌وجوی بهینه (Optimal BST)	۵.۴.۴
۱۳۴	مسئله‌ی کوله پشتی (Knapsack Problem)	۶.۴.۴
۱۴۰	روش حریصانه در طراحی الگوریتم‌ها	۵
۱۴۸	ویژگی‌های انتخاب حریصانه	۱.۵.۴
۱۴۹	انتخاب فعالیت‌ها	۲.۵.۴
۱۴۹	مسئله‌های کوله پشتی	۳.۵.۴
۱۵۰	مسائل زمان‌بندی	۴.۵.۴

۱۵۸	الگوریتم هافمن	۰.۵.۴
۱۵۹	الگوریتم حریصانه تقریبی برای مسئله بسته بندی	۷.۵.۴
۱۶۰	تمرین ها	۲.۰.۴

۱۶۱	روش های جست وجو	۶.۴
۱۶۱	روش بس گرد	۱.۶.۴
۱۶۲	درخت بازی	۲.۶.۴
۱۶۴	محدود کردن فضای جست وجو	۳.۶.۴

کلیه حقوق این جزو متعلق به دکتر محمد قدسی است و تکثیر و توزیع آن فقط با اجازه‌ی مؤلف مجاز است.

با تشکر فراوان از دانش‌جویان زیر که متن اولیه‌ی این جزو، را از کلاس‌های درس «روش‌های حل مسئله» و «ساختمان‌های داده‌ای و الگوریتم‌ها» در نیمسال‌های اول ۷۴-۷۵ و دوم ۷۶-۷۷ تهیه، تایپ و با حروف‌چینی گردند: طلا نفتشی، خشام فیلی، علی رضا ملک‌زاده، جلال سائبی برخیس، محمود رضا صلوانی بور، محمد سهدیان، افسانه نظری، ابوالفضل هادی اسفکر، حسنت فادری، مسعود اسدپور، سیدعلی اکبری، فرید آرش رستگار، محمود رضا حاصمی بور، روزبه پورنادر، آرش رجایان، سلام کاظمی، سبیت رسنی، ناصر عربی، احسان الیمعی، مهران سولیمانی کلامی، آزاده شاکری، فیضانی، ساسان دشتی نژاد و محمود رضا تهرانی، از دهاب سیرکنی هم به خاطر رسم تعدادی از شکل‌های این جزو تشکر می‌شود، این جزو، تاکنون بافارسیک حروف‌چینی شده است. از گروه بروزه‌ی فارسیک که این نرم‌افزار را رایگان در اختیار عموم قرار داده‌اند، ذیکر نشکر می‌نمایم.

فصل ۱

روش‌های تحلیل الگوریتم‌ها

هدف‌های تحلیل الگوریتم‌ها به شرح زیر است:

- بررسی رفتار الگوریتم قبل از پیاده‌سازی، از نظر زمان اجرا و مقدار حافظه‌ی مصرفی
- مقایسه‌ی الگوریتم‌ها از نظر کارایی

۱.۱ زمان اجرای الگوریتم‌ها

عوامل زیر در زمان اجرای یک برنامه موثرند:

- سرعت ساخت افزار،
- نوع کامپایلر،
- اندازه‌ی داده‌ی ورودی مسئله،
- ترکیب داده‌ای ورودی،
- پیچیدگی الگوریتم، و
- بارامترهای دیگر که تابع نابت در زمان اجرا دارند.

از این عوامل، سرعت ساخت افزار و نوع کامپایلر به صورت نابت در زمان اجرای برنامه‌ها موثر هستند. بارامترهای مهم، پیچیدگی الگوریتم است که تابعی از اندازه‌ی مسئله می‌باشد. ترکیب داده‌ی ورودی را نیز می‌توان با محاسبه‌ی پیچیدگی الگوریتم در بدترین حالت ورودی با درحال متوسط در نظر گرفت.

برای بررسی یک الگوریتم نابعی به نام $T(n)$ در نظر می‌گیریم که در آن n اندازه‌ی ورودی مسئله است. مسئله ممکن است شامل چند داده‌ی ورودی باشد. به عنوان مثال، اگر ورودی یک گراف باشد، علاوه بر تعداد رأس‌ها (n)، تعداد پالهای آن گراف (m) هم یکی از مشخصه‌های داده‌ی ورودی است. در این صورت زمان اجرای الگوریتم خواهد بود. ممکن است مسئله بین از چند بارامتر داشته باشد ولی برای تحلیل مسئله ناجاریم آن‌ها که

فصل ۱ روش‌های تحلیل الگوریتم‌ها

۲

مهم‌هستند در نظر بگیریم (عمل انتزاع^۱). نکته‌ی دیگر آن است که در تحلیل الگوریتم‌ها، مقادیر ثابت تابع چشم‌پوشی هستند. آن‌چه اهمیت دارد، سرعت رشد زمان اجرای برنامه نسبت به اندازه‌ی ورودی است.

روند تحلیل الگوریتم‌ها را با یک مثال شروع می‌کیم.

مثال: مرتب‌سازی مبتنی بر درج INSERTION-SORT

```

Insertion_Sort(A, n):
    Input: A (the array to sort)
           n (size of array)
1   for k := 2 to n
2       do key := A[k]
3           i := k - 1
4           while i > 0 and A[i] > key
5               do A[i+1] := A[i]
6               i := i - 1
7       A[i+1] := key

```

این الگوریتم در واقع طوری عمل می‌کند که در ابتدای مرحله‌ی k ام عناصر موجود در درایه‌های ۱ تا $k-1$ از این نسبت بهم مرتب هستند و در انتهای آن $A[k]$ در مکان مناسب بین این عناصر درج می‌شود. نحوه‌ی درج به این صورت است که $A[k]$ با تعویض با عنصر قبلی‌اش به سمت ابتدای آرایه حرکت می‌کند تا بالآخره دقیقاً در درایه‌ای که باید، متوقف می‌شود.

برای محاسبه‌ی زمان اجراء، ابتدا به هر یک از سطرهای برنامه زمان (هزینه‌ی) اجرای آن سطر را نسبت می‌دهیم و می‌شماریم که آن سطر چند بار تکرار می‌شود. این اعداد در جدول زیر مشاهده می‌شوند. تعداد تکرار سطر ۴ وابسته به داده‌ی ورودی دارد. تعداد این تکرار را در مرحله‌ی k ام c_k می‌نامیم. توجه کنید که در آن صورت سطر پنجم $1/k$ بار تکرار می‌شود.

خط	هزینه	تعداد
۱	c_1	n
۲	c_2	$n-1$
۳	c_3	$n-1$
۴	c_4	$\sum_{k=2}^n t_k$
۵	c_5	$\sum_{k=2}^n (t_k - 1)$
۶	c_6	$\sum_{k=2}^n (t_k - 1)$
۷	c_7	$n-1$

حاصل ضرب تعداد اتفاقات اجرای هر سطر و زمان اجرای آن را برای هر سطر به دست آورده و جمع می‌زنیم تا زمان اجرای کل بدست آید:

$$T(n) = c_1 n + (c_2 + c_3 + c_4)(n-1) + c_5 \sum_{k=1}^{n-1} t_k + (c_6 + c_7)(n-1)$$

مقدار ثابت‌ها به این ترتیب تابع الگوریتم افزایشی و جملن عواملی مستگی دارد و حتا در شرایط خاص به سختی

abstraction

۲.۱ بهترین و بدترین حالت و حالت متوسط

۳

قابل محاسبه‌اند، بنابراین ما هر بار به جای جمع چند ثابت و عباراتی از این قبیل، نایت جدیدی قرار می‌دهیم.

$$T(n) = An + B \sum_{k=1}^n t_k + C$$

۲.۱ بهترین و بدترین حالت و حالت متوسط

مقدار واقعی زمان اجرا به مقدار بدها بستگی دارد. «بهترین حالت»^۱ هنگامی اتفاق می‌افتد که آرایه از قابل مرتب شده باشد. در این حالت همواره داریم $t_1 = t_2 = \dots = t_n$ و

$$T(n) = An + B(n - 1) + C$$

که یک تابع خطی است، البته رفتار یک الگوریتم در بهترین حالت زیاد مهم نیست. آن‌چه مهم است رفتار الگوریتم در «بدترین حالت»^۲ و «حالت متوسط»^۳ است.

بیشترین زمان اجرای الگوریتم هنگامی اتفاق می‌افتد که آرایه برعکس مرتب باشد. در این صورت مقدار t_1 همواره برابر نباشد (مقایسه‌ی آخر با صفر هم حساب شده است). برای این حالت داریم:

$$\begin{aligned} T(n) &= An + B \sum_{k=1}^n t_k + C \\ &= An + B \left(\frac{(n-1)(n+2)}{2} - 1 \right) + C \\ &= an^2 + bn + c \end{aligned}$$

که یک تابع درجه‌ی دو بر حسب n است. با توجه به این که بهترین و بدترین رفتار الگوریتم متفاوتند، رفتار متوسط آن قابل توجه است.

حالت متوسط: اگر داده‌ی ورودی به صورت تصادفی داده شود رفتار الگوریتم (زمان اجرا) چه گونه است؟ (فرض می‌شود که همهی حالت‌های مختلف داده‌های ورودی، احتمال برابر دارند).

از جمله در مورد مثال فوق، $n!$ حالت (جای‌گشت) مختلف برای ترتیب n داده‌ی ورودی وجود دارد که به هر یک احتمال بیکسان و برابر $\frac{1}{n!}$ منسوب می‌شود. چیزی که در پس مفهوم حالت متوسط وجود دارد، این است که اگر چه ممکن است اجرا به اندازه‌ی زمان بدترین حالت طول بکشد، اما اگر دفعات زیادی الگوریتم بر ورودی‌های مختلف اعمال شود، به حد مذکور دست خواهیم یافت. و انتظار ما از زمان اجرا برای یک ورودی تصادفی چیست؟

best-case^۴
worst-case^۵
average-case^۶

۲

17

فصل ۱ روش‌های تحلیل الگوریتم‌ها

۴

زمان اجرا در حالت متوسط در مورد مثال فوق برابر است با

$$\bar{T}(n) = An + C + \frac{1}{n!} \sum_{i=1}^n \sum_{k=1}^n B t_{k,i}$$

(در این حالت ما عمل نکنک حالت‌ها را بررسی کرده و متوسط را یافته‌ایم). می‌توان ثابت کرد که کافی است به جای محاسبه‌ی مجموع فوق، از مقدار متوسط t_k یا \bar{t}_k استفاده کرد، یعنی

$$\bar{T}(n) = An + C + B \sum_{k=1}^n \bar{t}_k$$

برای محاسبه‌ی \bar{t}_k به مکانی که عنصر $A[k]$ باید در آن قرار بگیرد توجه می‌کیم که می‌تواند اولین درایه، دویمن درایه، ... یا همان درایه‌ی اولیه یعنی $t_{1,k}$ باشد. هر یک از این درایه‌ها احتمال وقوع برابر دارند که بهوضوح \bar{t}_k است. فرض کنید که در انتهای مرحله‌ی k ام در $[i] A[i:k]$ قرار بگیرد. در این حالت $t_k = k - i + 1$ (برای $i \geq 1$). پس داریم

$$\begin{aligned} \bar{t}_k &= \sum_{i=1}^k \frac{1}{k} (k - i + 1) \\ &= \frac{1}{k} \times \frac{k(k+1)}{2} \\ &= \frac{k+1}{2} \end{aligned}$$

پس می‌توان گفت که هر بار عضو آخر به طور متوسط به وسیله آرایه منتقل می‌شود. و نیز داریم

$$\begin{aligned} \bar{T}(n) &= An + C + B \sum_{k=1}^n \frac{k+1}{2} \\ &= an^2 + bn + c \end{aligned}$$

پس رفتار حالت متوسط و بدترین حالت بکسان است.

۳.۱ مرتبه‌ی الگوریتم‌ها

مسئولاً بدترین حالت اهمیتی ندارد. بلکه الگوریتم‌ها از نظر بدترین حالت و حالت متوسط بررسی می‌شوند. وقتی حالت متوسط و بدترین حالت مختلف باشند، وقت خاصی لازم است. برای مثال در مورد quicksort، بدترین حالت $O(n^2)$ و حالت متوسط $O(n \log n)$ است که وقتی n عدد بزرگی باشد، تفاوت عددیای کاربردی بین آن‌ها بسیار است. آن‌چه برای ما مهم است، رفتار الگوریتم در اندازه‌ی وروزی بزرگ یا خیلی بزرگ است از این حالت. جنان‌چه خواهیم دید، ثابت‌هایی چون c_1, c_2, \dots, c_n اهمیتی ندارند، و این با مفهوم «پیچیدگی الگوریتم» نشان داده می‌شود. برای درک بهتر از مفهوم پیچیدگی، فرض کنید برای حل یک مسئله جهار الگوریتم داشتیم که بیرروی n عدد کار می‌کنند. برای این الگوریتم‌ها $T(n)$ را به طور دقیق بدست آورده‌ایم که در جدول ۱ انتساب داده شده‌اند. جدول فوق نشان می‌دهد که برای n های کوچک الگوریتم c_1, c_2, \dots, c_n بیشتر از الگوریتم c_{n+1}, \dots, c_{2n} است. هم‌چنین می‌توان دید که برای الگوریتم‌های با درجه‌ی پیچیدگی زیاد، افزایش سرعت مماثلت تابع چندانی ندارد. این موضوع را می‌توان در منحنی‌های زمان اجرای این الگوریتم‌ها برای n های مختلف در شکل ۱.۱ دید.

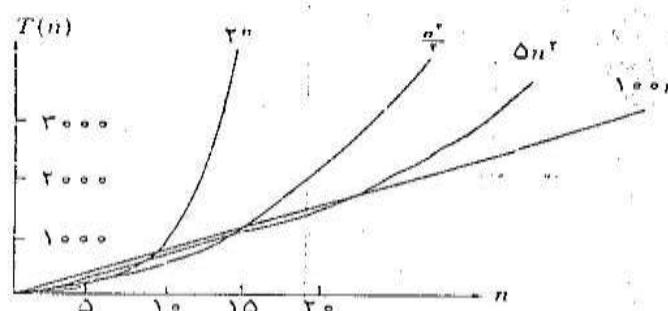
complexity of algorithm^۳



۱.۴ تابع‌های رشد

الگوریتم	$T(n)$	$O(\cdot)$	مسئله که در حل می‌شود	حداکثر اندازه‌ی ۱۰۰۰ ثانیه برروی ۱۰۰۰ ماشین قابل حل	حداکثر اندازه‌ی ۱۰۰۰ ثانیه برروی ۱۰۰۰ ماشین	نسبت	نسبت	برابر سریع‌تر انتخاب کرد و باشیم	نسبت آنکه ماشینی ۱۰۰۰ برابر سریع‌تر انتخاب کرد
n^2	$\Theta(n^2)$	$100n$	۱۰	۱۰۰	۱۰	۱۰	۱۰	۱۰	۱۰
n^3	$\Theta(n^3)$	$5n^2$	۲۵	۴۵	۱۴	۳۲	۱۳	۱۳۱/۹۴	۱۳۱/۹۴
n^4	$\Theta(n^4)$	$n^3/2$	۲۷	۷۲	۱۲	۲۴	۲۷	۱۲۰/۹۹	۱۲۰/۹۹
2^n	$\Theta(2^n)$	2^n	۱۳	۱۰	۱۰	۱۳	۱۰	۲۰	۲۰

جدول ۱.۱: مقایسه‌ی پیچیدگی الگوریتم‌ها



شکل ۱.۱: زمان‌های اجرای چهار الگوریتم برای یک مسئله.

از این مثال در می‌بایس که الگوریتم n^2 برای n ‌های کوچک هم سریع‌تر است و هم آسان‌تر بسازی می‌شود، اما برای «های بزرگ‌تر» الگوریتم در عمل می‌استفاده می‌شود، برای رفع این مشکل، عدددهای «کوچک‌ر» را در نظر نمی‌گیریم و ملاک مقایسه را «های بزرگ‌تر می‌گیریم، نکته‌ی بسیار مهم این است که همواره عددی مثل «۱۰۰۰» خواهد شد به طوری که برای مقادی $n > 100$ ، منحنی زمان اجرای الگوریتم n^2 از الگوریتم $10^{10}n$ جلو بزند.

۱.۴.۱ تابع‌های رشد

برای بررسی مرتبه‌ی الگوریتم‌ها تابع‌های رشد (growth functions) استفاده می‌کیم که با نمادهای O , Θ , Ω , \mathcal{O} و Ω نمایش داده می‌شوند. از جمله گزاره‌هایی که به چشم می‌خورد، جنین است:

- بدترین حالت الگوریتم مرتب‌ساز متنی بر درج، $T(n) = \Theta(n^2)$ یعنی از مرتبه‌ی دقیق n^2 است، این الگوریتم در حالت متوسط نیز $\Theta(n^2)$ است.
- الگوریتم HeapSort از $O(n \log n)$ یا مرتبه‌ی $n \log n$ است.
- کلیه‌ی الگوریتم‌های مرتب‌ساز مبتنی بر مقایسه، از مرتبه‌ی $\Omega(n \log n)$ هستند.

تعريف تابع‌های رشد

برای (n) داده شده، $(n) \Theta(g(n))$ را به شکل مجموعه‌ی توابع زیر تعریف می‌کیم:

فصل ۱ روش‌های تحلیل الگوریتم‌ها

$$\Theta(g(n)) = \{f(n) : \exists c_1, c_2 > 0 \text{ and } n_0 \text{ such that } \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

عبارت $c_1 g(n) \leq f(n) \leq c_2 g(n)$ به این معنی است که درجهٔ رشد f و g بسان است. برای نشان دادن

دقيق این مفهوم، می‌نویسیم

$$f(n) \in \Theta(g(n))$$

و یا به شکل ساده‌تر

$$f(n) = \Theta(g(n))$$

مثلًا می‌دانیم که در مورد الگوریتم مرتب‌ساز مبتنی بر درج، $T(n) = \Theta(n^r)$. زیرا می‌توان c_1 و c_2 را طوری یافت که

$$c_1 n^r \leq an^r + bn + c \leq c_2 n^r$$

مثال: ثابت کنید که $100n^r + 5n - 4 = \Theta(n^r)$

روشن می‌باشد که اگر $c_1 = 1$ و $c_2 = 200$ باشد، $c_1 n^r \leq 100n^r + 5n - 4 \leq c_2 n^r$ داریم؛

می‌توان ثابت کرد که هر چند جمله‌ای از مرتبهٔ r دقيق اولین جمله‌اش است، یعنی

$$a_k n^k + a_{k-1} n^{k-1} + \dots = \Theta(n^k)$$

در مورد مرتب‌سازی مبتنی بر درج داریم:

$$\begin{aligned} T(n) &= \Theta(n^r) \\ &= \Theta(100n^r) \\ &\neq \Theta(n^r) \\ &\neq \Theta(n^r \log n) \\ &\neq \Theta(n \log n) \end{aligned}$$

برای اثبات $T(n) \neq \Theta(n \lg n)$ می‌توان نشان داد که نمی‌توان ثابت‌های c_1 و c_2 را پیدا کرد که برای $n > n_0$

داشته باشیم:

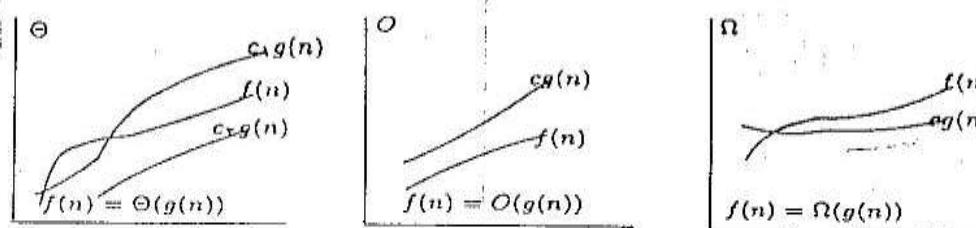
$$c_1 n \lg n \leq n^r \leq c_2 n \lg n$$

تعریف: برای $g(n)$ داده شده، $\mathcal{O}(g(n))$ را به شکل مجموعه‌ای توابع زیر تعریف می‌کنیم:

$$\mathcal{O}(g(n)) = \{f(n) : \exists c > 0 \text{ and } n_0 \text{ such that } \forall n \geq n_0, c \leq f(n) \leq cg(n)\}$$

مثلًا در مورد مرتب‌ساز مبتنی بر درج، داریم

۲.۱ تابع‌های رشد



شکل ۲.۱: زمان‌های اجرای چهار الگوریتم برای یک مسئله.

$$\begin{aligned}
 T(n) &= O(n^r) \\
 &= O(10 \cdot n^r) \\
 &\neq O(n^r) \\
 &\neq O(n^r \lg n) \\
 &\neq O(n \lg n) \\
 &\neq O(n)
 \end{aligned}$$

این نکته جالب است که می‌توان گفت مسئله از $O(n^{100})$ است، ولی این اطلاع چندان مفید نیست. به همین جهت باید در حالت‌هایی که محاسبه‌ی Θ مشکل است، باید تابع داخل پرانتز O را تا حد امکان گوچک نر کیم.

تعریف: برای $(g(n))$ داده شده، $\Omega(g(n))$ را به شکل مجموعه‌ی ثوابع زیر تعریف می‌کنیم:

$$\Omega(g(n)) = \{f(n) : \exists c > 0 \text{ and } n_0 \text{ such that } \forall n \geq n_0, c \cdot g(n) \leq f(n)\}$$

مثلثاً برای مرتب‌سازی مبتنی بر درج داریم

$$\begin{aligned}
 T(n) &= \Omega(n \log n) \\
 &= \Omega(n) \\
 &= \Omega(n^r) \\
 &\neq \Omega(n^r \log n)
 \end{aligned}$$

در عمل نماد Ω برای نشان دادن حد پایین یک تابع دیگر به کار می‌رود.
تابع Θ در واقع عطف O و Ω است به معنی

$$f(n) = \Theta(g(n)) \Leftrightarrow f(n) = O(g(n)) \wedge f(n) = \Omega(g(n))$$

فصل ۱ روش‌های تحلیل الگوریتم‌ها

مفهوم تابع‌های رشد تعریف شده در شکل ۲.۱ نشان داده شده است.
دو نساد زیر نیز در همین رابطه تعریف شده‌اند:

تعریف:

$o(g(n)) = \{f(n) \mid \text{for any positive constant } c > 0, \exists n_0 > 0, \text{ such that } \forall n > n_0 : 0 \leq f(n) < cg(n)\}$

این رابطه مزدیگی زیادی با O دارد بالین تفاوت که دیگر $f(n)$ نمی‌تواند با $c g(n)$ برابر باشد و باید الرام‌آ کوچکتر باشد.

تعریف:

$\omega(g(n)) = \{f(n) \mid \text{for any positive constant } c > 0, \exists n_0 > 0, \text{ such that } 0 \leq cg(n) < f(n)\}$

رابطه‌ی Ω مثلاً Ω مثل o با O است.

نتیجه‌ای که از تعریف‌های فوق به دست می‌آید را در عبارت‌های زیر (هر چند نادقيق از نظر ریاضی) خلاصه می‌کیم:

اگر f درجه‌ی رشد F و g درجه‌ی رشد G باشد.

$F = \Theta(g)$	$\iff f = g$
$F = O(g)$	$\iff f \leq g$
$F = o(g)$	$\iff f < g$
$F = \Omega(g)$	$\iff f \geq g$
$F = \omega(g)$	$\iff f > g$

مثالی دیگر

الگوریتم مرتب‌سازی مبتنی بر ادغام (MergeSort) را در نظر می‌گیریم.

```
Merge_Sort(A, p, r):
    Input: A (the array to sort)
           p (starting index)
           r (ending index)
1   if p < r
2       then q := (p + r) div 2
3           Merge_Sort(A, p, q)
4           Merge_Sort(A, q + 1, r)
5           Merge(A, p, q, r)
```

A

(23)

۵. روش‌های تحلیل الگوریتم‌ها

این الگوریتم، هر بار آن بخش از آرایه را که می‌خواهد مرتب کند، ابتدانصف می‌کند. هر نیمه را به صورت بازگشتنی مرتب می‌کند و با ادغام دو بخش مرتب شده، کار را تمام می‌کند.

برای ادغام دو لیست مرتب، ساده‌ترین الگوریتم به این شکل است که دو اشاره‌گر را باید بر ابتدای لیست‌ها فرار داد و هر بار عدد کمتر را به لیست خروجی افزود. اگر طول دو لیست اول m و n باشد، زمان مسئله $O(m + n)$ است. منتها این مسئله از $O(m + n)$ فضای اضافی می‌برد.

ادغام کردن «درجا»، یا in-place merging نوعی ادغام است که از فضای اضافی برای ادغام استفاده نمی‌کند. در واقع فضای اضافی مورد نیاز از $O(1)$ می‌شود.

اثبات درستی الگوریتم، به شکل استقرایی صورت می‌گیرد، یعنی برای پایه با $n = 1$ اثبات مسئله بدیهی است و برای $n > 1$ درستی آن به سادگی از فرض استقرار اثبات می‌شود (فرض می‌کنیم برای $n/2$ اثبات صورت گرفته است).

برای این الگوریتم داریم:

$$T(n) = \begin{cases} a & n = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + bn & n > 1 \end{cases}$$

که می‌توان (بعداً می‌بینیم) ثابت کرد که

$$T(n) = \Theta(n \log n).$$

۵.۱ روش‌های تحلیل الگوریتم‌ها

الگوریتم‌ها به دو دسته‌ی اصلی تقسیم می‌شوند: الگوریتم‌های ترتیبی و بازگشتنی.

۵.۱.۱ الگوریتم‌های ترتیبی

برای بدست آوردن زمان اجرای یک الگوریتم ترتیبی فرض می‌کنیم که اجرای یک دستور ساده (Statement) زمان ثابتی نیاز دارد. بنابراین اگر تعداد ثابتی دستور ساده پشت سرهم باشد، در مجموع زمانی ثابت می‌گیرند $T(n)$ زمان اجرای یک تکه برنامه به صورت‌های زیر محاسبه می‌شود:

- اگر تکه برنامه مورد نظر خود شامل k تکه برنامه با زمان‌های احرای $f_1(n), f_2(n), \dots, f_k(n)$ باشد، در آن صورت $T_k(n) = O(f_1(n) + f_2(n) + \dots + f_k(n))$

$$T(n) = \sum_{k=1}^k T_k(n) = O(\max_{1 \leq i \leq k} (f_i(n)))$$

حال (نحوه) می‌تواند اصل حینه‌دهنده برنامه = (زیرنظام از زیرنظام) را داشته باشد.

- اگر تکه برنامه ساختار حلقه داشته باشد (شامل for, while, repeat و امثال آن) و n بار تکه برنامه‌ای دیگر با زمان اجرای $f(n)$ بازگردد، باز $S(n) = O(f(n) \cdot n)$ را تکرار کند.

بعض از مواقع حلقة `for` یا `while` هم ساخته می‌شود که تشخیص این نواره نیاز به دقت دارد.

$$T(n) = g(n)S(n) = O(g(n)f(n))$$

- اگر ساختار if باشد و قسمت‌های else آن به ترتیب زمان‌های $T_{\text{v}}(n)$ و $T_{\text{r}}(n) = O(f_{\text{v}}(n))$ باشند، $T_{\text{v}}(n) = O(f_{\text{v}}(n))$

$$T(n) = \max\{T_{\text{v}}(n), T_{\text{r}}(n)\} = O(\max\{f_{\text{v}}(n), f_{\text{r}}(n)\})$$

به طور شهودی، مرتبه‌ی زمان اجرای یک الگوریتم، همان مرتبه‌ی تکمای است که در برنامه بیشترین زمان را می‌گیرد. چرا که ما همیشه برای تابع رشد بدترین حالت را در نظر می‌گیریم.
به این مثال برای الگوریتم Bubble Sort توجه کنید:

```

For i:=1 to n-1 do
  For j:=n downto i+1 do
    If A[j] > A[j-1]
      Then Swap ( a[j] , A[j-1] )
  
```

توجه کنید که زمان مقایسه و تعویض را ثابت در نظر می‌گیریم. تعداد دقیق مقایسه‌ها و حداقل تعداد تعویض‌ها برابر است با

$$T(n) = \sum_{i=1}^n \sum_{j=i+1}^n O(1) = \Theta(n^2)$$

این الگوریتم در بدترین حالت برسی شده است.^۲ البته یا توجه به مطالب درس همین موضوع را بدون محاسبه می‌توان فهمید، چون زمان مقایسه و تعویض را ثابت در نظر گرفتیم، زمان if یک عدد ثابت است. تکرار حلقه داخلی در بدترین حالت از مرتبه‌ی n است و بنابراین کل آن از نیز از مرتبه n می‌شود. تکرار حلقه پرروشی نیز در بدترین حالت از مرتبه n است و چون کل حلقه‌ی داخلی از مرتبه‌ی n بود، بنا به قوانین گفته شده زمان کل حلقه از مرتبه‌ی حاصل ضرب آن دو بعنی n^2 است.

۶.۱ الگوریتم‌های بازگشتی

معمولاً الگوریتم‌های بازگشتی مسئله را به دو (یا چند) زیرمسئله‌ی کوچکتر تقسیم می‌کند و آن‌ها را به صورت بازگشتی و با همان الگوریتم حل و نتایج آن‌ها با با هم ترکیب می‌کند. بنابراین زمان اجرای راه حل، حاصل جمع زمان حل زیرمسئله‌ها به علاوه‌ی زمان لازم برای شکستن مسئله به زیرمسئله و نیز ترکیب جواب‌های آن‌هاست.

^۲ پدیدست آوردن تعداد تعویض‌ها در حالت متوسط کار چندان ساده‌ای نیست و نیاز به بحث آماری دارد.

۱.۷.۱ مسئله‌ی برج‌های هانوی

جداًقل تعداد حرکت در این مسئله وقتی است که بزرگترین سکه مستقیماً به میله‌ی مقصد منتقل شود، برای این کار لازم است که بقیه‌ی سکه‌های دوم باشند که این کار به صورت بازگشتی انجام می‌شود، پس از انتقال سکه‌ی بزرگ، بقیه‌ی سکه‌ها ($1 - n$ عدد) به صورت بازگشتی به میله‌ی مقصد منتقل می‌شوند، توجه کنید که با این الگوریتم کلیدی قواعد بازی رعایت شده و نمی‌توان الگوریتمی با تعداد حرکت کمتر برای این مسئله پیدا کرد، با این توضیح، اگر $T(n)$ کمینه‌ی تعداد حرکت‌ها برای «سکه باشد، داریم

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

که این یک «رابطه‌ی بازگشتی» (Recurrence Relation) است.
برای مثال ذیگر به زمان الگوریتم مرتب‌ساز مبتنی بر ادغام توجه کنید:

$$T(n) = \begin{cases} 0, & n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + Cn, & n > 1 \end{cases}$$

اگر $n = 2^k$ داریم،

$$T(n) = \begin{cases} 0, & n = 1 \\ T(n) = 2T(n/2) + Cn, & n > 1 \end{cases}$$

در این مثال Cn هزینه ادغام است و $T(\lceil n/2 \rceil)$ و $T(\lfloor n/2 \rfloor)$ هرینه‌های حل زیر مسئله‌ها بنای استخراج هستند.

۱.۸ روش‌های حل رابطه‌های بازگشتی

رابطه‌های بازگشتی را می‌توان به روش‌های زیر حل کرد.

۱. استقرار: خدوس خوب و استقرار

۲. تکرار با جایگذاری: بازگردان فرمول و بعدست آوردن جواب به طور صریح

۳. قضیه‌ی اصلی

۴. روش‌های حل رابطه‌های بازگشتی همگن و ناممگن

۱.۷.۲ خدوس و استقرار

در این روش ابتدا سعی می‌کنیم جواب را خدوس بزنیم و در مرحله‌ی بعد آن را اثبات می‌کنیم، برای خدوس جواب ممکن است از قضیه‌ی اصلی استفاده کیم.

اگر بخواهیم اثبات کنیم که $T(n) = O(f(n))$ باشد نااستقرار اثبات کنیم که $T(n) \leq Cf(n)$ باشد. از دلایل این استقرار نشان دهیم که بحداقل یک مقدار ثابت برای c وجود دارد که در این نامعادله صدق کند. برای اثبات $T(n) = \Omega(f(n))$ باید به همین ترتیب رابطه‌ی $T(n) \geq cf(n)$ و برای اثبات $T(n) = \Theta(f(n))$ باید هم $T(n) = \Omega(f(n))$ و هم $T(n) = O(f(n))$ را اثبات کنیم.

مثال ۱

$$\begin{aligned} T(n) &= 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n \\ T(2) &= T(1) = O(1) \end{aligned}$$

نکته: در حل رابطه‌های بازگشتنی برای بدست آوردن مرتبه با $O(\dots)$ از \dots می‌توان صرف نظر کرد. البته در صورتی‌که از این علایم صرف نظر نکنیم جواب دقیق‌تری بدست خواهیم آورد.
 حل مثال مانوچه به این که هر بار مسئله را به دو زیر مسئله با اندازه‌ی نصف تقسیم می‌کنیم. بعد از n بار مسئله‌ای ساده‌ای واحد خواهیم داشت و چون زیر مسئله‌ها را با (n) باهم ترکیب می‌کنیم، حدس می‌زنیم که کل مسئله از مرتبه‌ی $O(n \lg n)$ است. باید ثابت کنیم که عدد مثبت c وجود دارد به طوری که $T(n) \leq cn \lg n$ باشد. اثبات:

$$n = 2 \Rightarrow T(2) \leq 2c \Rightarrow c \geq \frac{T(2)}{2} > 0$$

فرض استخراج: برای $k < n$ فرض می‌کنیم $T(k) \leq ck \lg k$
 حکم استخراج: باید ثابت کنیم $T(n) \leq cn \lg n$. داریم:

$$\begin{aligned} T(n) &\leq 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n \\ &\leq 2ck \lg \left\lfloor \frac{n}{2} \right\rfloor + n \\ &= cn \lg n - cn + n \\ &\leq cn \lg n \end{aligned}$$

اگر $1 \leq n$ حکم اثبات می‌شود. نکته‌ی مهم در این بروز حل آن ایست که باید در حکم استخراج دقیقاً به همان فرمولی برسیم که حدس زداییم (متلا $cn \lg n$). در صورتی‌که این امر اثبات نشود، باید حدس اولیه را تصویب کرد.

مثال ۲

برای مثال نیکو به تحلیل مرتب‌سازی منتهی بر ادعای بوجه کنید. فرض می‌کنیم $n = 2^k$.

$$\begin{aligned} T(1) &= n \\ T(n) &= 2T\left(\frac{n}{2}\right) + bn \end{aligned}$$

حدس: $T(n) \leq cn \lg n = T(n) = O(n \lg n)$

باشد: $T(2) = a$

فرض استخراج: برای $k \geq 1$ داریم $T(n/2^k) \leq c n/2^k \lg n/2^k$ آن‌گاه

۱.۲ روش‌های حل رابطه‌های بازگشتی

$$\begin{aligned} T(n) &\leq 2c\frac{n}{\gamma} \lg \frac{n}{\gamma} + bn \\ &= cn \lg n + (b - c)n \leq cn \lg n \\ &\leq cn \log n, \text{ if } b - c \leq 0 \text{ and } c \geq a/\gamma \end{aligned}$$

این اثبات فقط بیان می‌دارد که به ازای $n = 2^k$ الگوریتم از $O(n \lg n)$ است اما توجه به این نکته ضروری است که برای هر عدد دلخواه n دو توان متوالی دو را می‌توان یافت که عدد مذکور بین آن دو باشد ($2^{k+1} \leq n < 2^{k+2}$). چون تابع رشد تابعی صعودی است می‌توان اثبات کرد که $T(n)$. مرتبه‌ی الگوریتم برای n همان مرتبه‌ی $T(2^k)$ است.

تمرین: برای تابع

$$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor + 1) + n, \quad n > 100 \\ T(1) &= T(2) = \dots = T(100) = 1 \end{aligned}$$

حدس می‌زنیم: $T(n) = O(n \lg n)$ (در واقع $T(n) \leq cn \lg n$) این حدس را ثابت کنید.

مثال ۳

$$T(n) = T(\lfloor \frac{n}{\gamma} \rfloor) + T(\lceil \frac{n}{\gamma} \rceil) + 1$$

حدس می‌زنیم تابع رشد از مرتبه (n) است. باید ثابت کنیم $T(n) \leq cn$. اگر از استقرآ برای اثبات استفاده کنیم، در نهایت خواهیم داشت،

$$T(n) \leq c\lfloor \frac{n}{\gamma} \rfloor + c\lceil \frac{n}{\gamma} \rceil + 1 = cn + 1 \geq Cn$$

بنابراین حدس اشتباه است که به این صورت آن را ناصحیح می‌کنیم: $T(n) \leq cn + b$. حال داریم:

$$T(n) \leq c\lfloor \frac{n}{\gamma} \rfloor + c\lceil \frac{n}{\gamma} \rceil + 2b + 1 = cn + 2b + 1 \leq Cn + B \Rightarrow B \leq -1$$

که اثبات کامل می‌شود.

مثال ۴

$$T(n) = 2T(\lfloor \frac{n}{\gamma} \rfloor) + n$$

حدس می‌زنیم تابع رشد از مرتبه n باشد. باید $T(n) \leq cn$ باشد. با استفاده از استقرآ داریم،

$$T(n) \leq 2C\lfloor \frac{n}{\gamma} \rfloor + n \leq cn + n = (c + 1)n \geq Cn$$

در مورد این مثال حدس اولیه غلط است. ولی از برای نتیجه‌ی به دست آمده متوجه می‌شویم که مرتبه‌ی الگوریتم بالاتر از n است. حدس جدید $T(n) \leq cn \log n + b$ را اگر امتحان کنیم درست در می‌آید. بنابراین از راه‌های حل بعضی از رابطه‌های بازگشتی تغییر متغیر است. به مثال زیر توجه کنید.

مثال ۵

$$T(n) = 2T(\sqrt{n}) + \log n$$

متغیر جدید m را به صورت $m = \log n$ در نظر می‌گیریم. در نتیجه خواهیم داشت، $T(2^m) = 2T(2^{\frac{m}{2}}) + m$. و یا

$$S(m) = 2S\left(\frac{m}{2}\right) + m$$

با توجه به مثال‌های قبلی داریم.

$$S(m) = O(m \log m) \Rightarrow T(n) = O(\log n \log \log n)$$

مثال ۶

می‌خواهیم با کمترین تعداد مقایسه‌های بین عناصر یک آرایه‌ی A ، کوچکترین و بزرگترین عدد آرایه را به دست آوریم. می‌توان اثبات کرد که کمترین تعداد مقایسه‌های ممکن $2 - \lceil \frac{n}{2} \rceil$ است.^۸

روش غیر بازگشتی اول: وقتی بزرگترین عنصر به دست آمد نیاز به مقایسه‌ی آن با کوچکترین عنصری که تا کنون به دست آمده نیست. پس از کردن بزرگترین n عنصر $1 - n$ مقایسه لازم دارد. برای محاسبه‌ی کوچکترین عنصر $2 - n$ مقایسه دیگر کافی است. پس این کار با $2 - 2n$ مقایسه ممکن است.

روش غیر بازگشتی دوم: برای $n = 2k$ با یک مقایسه بین هر دو عنصر متولی Min ‌ها و Max ‌های آن‌ها را پیدا می‌کنیم. با مقایسه‌ی همه‌ی Min ‌ها کوچکترین عنصر و با مقایسه‌ی همه‌ی Max ‌ها بزرگترین عنصر را به دست می‌آوریم. این روش برای $n = 2k$ بهینه است و به $2 - 2 - n/2 + (n/2 - 1) + (n/2 - 1) = 3n/2$ مقایسه نیاز دارد. تعداد مقایسه‌های این روش برای $1 - 2k + 2k = 3n/2$ است و بهینه نیست.

در روش بازگشتی آرایه را به دو قسمت تقریباً مساوی تقسیم می‌کنیم. در هر قسمت Min و Max را به دست می‌آوریم. سپس دو Min را با هم و دو Max را با هم مقایسه می‌کنیم تا بزرگترین و کوچکترین عنصر به دست آید.

^۸ اثبات این مطلب خیلی ساده است!

```

Procedure Maximum (A: array of integer; head, tail : integer;
                  var max, min : integer);
var
  max2, min2 : integer;
begin
  if head = tail then begin
    max := A[head];
    min := max;
  end else if tail - head = 1 then begin
    if A[head] > A[tail] then begin
      max := A[head];
      min := A[tail];
    end else begin
      max := A[tail];
      min := A[head];
    end;
  end else begin
    Maximum(A, head, (head+tail) div 2, max, min);
    Maximum(A, (head+tail) div 2+1, tail, max2, min2);
    if max2 > max then max := max2;
    if min2 < min then min := min2;
  end;
end;

```

اگر تعداد مقایسه‌های عنصرهای آرایه در این روش $T(n)$ باشد، داریم.

$$T(n) = \begin{cases} 0 & n = 1 \\ 1 & n = 2 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 2 & n > 2 \end{cases}$$

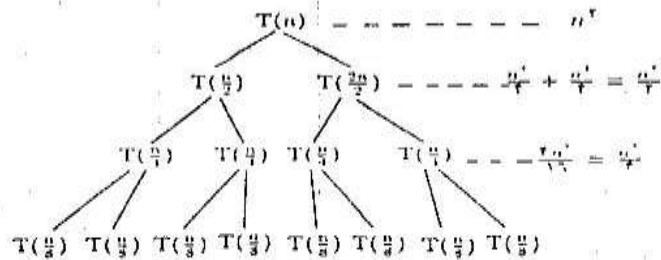
این راه حل بزرگتر نیست، چون مثلاً $T(8) = 6$ در حالی که می‌توان با ۷ مقایسه $(2 - \frac{18}{3})$ این کار را انجام داد. البته راه حل فوق برای $n = 2^k$ بینه است و با استفاده از استقرا بوسادگی می‌توان آن را نشان داد.

$$T(2^k) = 2[2 \times 2^{k-1} - 2] + 2 = 3 \times 2^{k-1} - 2$$

راه حل زیر برای این مسئله بینه است. آرایه را به دو قسمت با اندازه‌ی ۲ و $2 - 2$ تقسیم کن. با $1 + T(n-2)$ صورت،

$$T(n) = \begin{cases} 0 & n = 1 \\ 1 & n = 2 \\ T(n-2) + 2 & n > 2 \end{cases}$$

که جواب آن همان مقدار بینه است.



شکل ۳.۱: درخت بازگشت برای مثال ۱

۲.۷.۱ تکرار با جای‌گذاری

در این روش رابطه‌ی بازگشتی را آن قدر بسط می‌دهیم تا به جواب نهایی برسیم.

$$\text{مثال ۱: } T(n) = \tau T(\lfloor \frac{n}{\tau} \rfloor) + n$$

$$\begin{aligned} T(n) &= \tau T(\lfloor \frac{n}{\tau} \rfloor) + n \\ &= \tau^2 T(\lfloor \frac{\lfloor \frac{n}{\tau} \rfloor}{\tau} \rfloor) + \tau \lfloor \frac{n}{\tau} \rfloor + n = \tau^2 T(\lfloor \frac{n}{\tau^2} \rfloor) + \tau \lfloor \frac{n}{\tau} \rfloor + n \\ &= \tau^3 T(\lfloor \frac{n}{\tau^3} \rfloor) + \tau^2 \lfloor \frac{n}{\tau^2} \rfloor + \tau \lfloor \frac{n}{\tau} \rfloor + n \\ &= \dots \\ &= \dots \\ &\leq \tau^i T(\frac{n}{\tau^i}) + n \sum_{j=0}^{i-1} (\frac{n}{\tau^i})^j \end{aligned}$$

از برابری $\lfloor \frac{n}{\tau^i} \rfloor = \frac{n}{\tau^i}$ در فرمول‌های فوق استفاده می‌کیم.
برای حل رابطه باید آن را باز کیم تا به $T(1)$ برسیم. داریم $\frac{n}{\tau^i} = 1 \Rightarrow i = \log_\tau n$ ، بنابراین

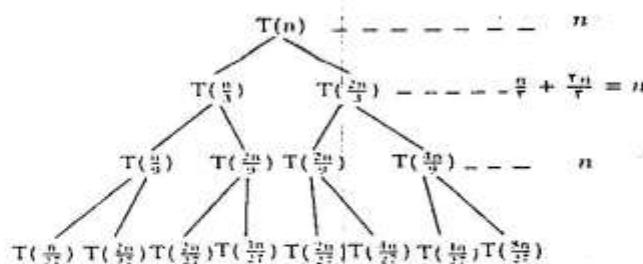
$$T(n) \leq \tau^{\log_\tau n} \cdot T(1) + n \sum_{j=0}^{\log_\tau n - 1} (\frac{n}{\tau})^j$$

برای n های بزرگ این مجموع مقداری ثابت است. لذا $T(n) \leq C_1 n^{\log_\tau \tau} + C_2 n$. اما می‌دانیم که

$$\lim_{n \rightarrow \infty} \sum_{j=0}^{\log_\tau n - 1} (\frac{n}{\tau})^j = \tau \Rightarrow C_1 \sim \tau$$

و داریم $\tau^{\log_\tau n} = n^{\log_\tau \tau}$. لذا

$$T(n) \leq C n^{\log_\tau \tau} + \tau n \Rightarrow T(n) = O(n)$$



شکل ۱.۲: درخت بازگشت برای مثال ۲

در اکثر موردها روش بسط فرمول جواب می‌دهد ولی در بعضی حالات نمی‌توان از باز کردن فرمول به جایی رسید. مثلاً، رابطه‌ی بازگشتی برای تولید عددان فیبوناچی از این نوع است: $F(n) = F(n - 1) + F(n - 2)$ که داریم $F(1) = F(2)$. برای حل رابطه‌های مانند این، از روش حل روابط بازگشتی همگن استفاده می‌شود.

درخت بازگشت

بکی از روش‌های خوب برای حل با حدس رابطه‌های بازگشتی از طریق تکرار استفاده از درخت بازگشت است.

$$T(n) = 2T\left(\frac{n}{3}\right) + n^r \quad \text{مثال ۱}$$

$$\begin{aligned} T(n) &= n^r + \frac{n^r}{3} + \dots + \frac{n^r}{3^k} + \dots + \frac{n^r}{3^{m-k}} \\ &= n^r \left(1 + \frac{1}{3} + \dots + \frac{1}{3^k} + \dots + \frac{1}{3^{m-k}}\right) \\ &\leq 3n^r \\ \Rightarrow T(n) &= O(n^r) \end{aligned}$$

توجه: آخرین سطح این درخت دارای "برگ" با مقدارهای ۱ خواهد بود.

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{5n}{8}\right) + n \quad \text{مثال ۲}$$

این درخت به گونه‌ای رشد می‌کند که در نهایت بعد از k مرحله به $T(1)$ می‌رسیم. در اینجا k همان ارتفاع بیشینه‌ی درخت است. این درخت متوازن نیست و در سمت چپ ارتفاع کمتری دارد. برای بدست آوردن ارتفاع بیشینه‌ی درخت، داریم، $1 = \log_2 n = i = \log_{\frac{5}{8}} n$ ، پس $i = \log_{\frac{5}{8}} n$ ، لذا $T(n) \leq n \log_{\frac{5}{8}} n$. در تبعیه $\log_{\frac{5}{8}} n = O(\log_2 n)$.

۳.۷.۱ قضیه‌ی اصلی

در اینجا ما خود قضیه‌ی اصلی را اثبات نمی‌کنیم ولی می‌توان آن را از روش جایگذاری اثبات کرد.
برای $1 \leq n \leq 10$ حل رابطه‌ی بازگشتی $f(n) = nT\left(\frac{n}{k}\right) + f(n)$ (که در آن $\frac{n}{k}$ می‌تواند به صورت کافی با سقف باشد) بدغیرت زیر است:

الف - اگر $f(n) = O(n^{\log_k n})$ ، برای $n > 10$ (معنی رشد نابغه $f(n)$ از نابغه $n^{\log_k n}$ به صورت چند جمله‌ای کمتر مانند). در این صورت، $T(n) = \Theta(n^{\log_k n})$.

ب - اگر $f(n) = \Theta(n^{\log_k n} \log_r n)$ ، در این صورت،

ج - اگر $f(n) = \Omega(n^{\log_k n})$ ، در این صورت، $T(n) = \Theta(f(n))$.

بعبارت ساده تر ولی شادقین، قضیه‌ی اصلی را می‌توان به صورت زیر بیان کرد. اگر G درجه‌ی رشد نابغه $f(n) = n^{\log_k n}$ و F درجه‌ی رشد نابغه $f(n)$ باشد، خواهیم داشت:

$$T(n) = \Theta(f(n)), F > G$$

$$T(n) = \Theta(g(n)), G > F$$

$$T(n) = \Theta(g(n) \log_r n), F = G$$

نحوه: $>$ به معنی بزرگتر یا کوچکتر چند جمله‌ای است. در صورتی که هیچ کدام از حالت‌های بالا وجود نداشته باشد نمی‌توان قضیه‌ی اصلی را اعمال کرد و باید از روش‌های دیگر استفاده کرد.

مثال ۱: $T(n) = 4T\left(\frac{n}{4}\right) + n$

داریم، n و $f(n) = n^{\log_4 4} = n^1 = n$. بدینهی است که درجه‌ی رشد n^1 از n به صورت چند جمله‌ای بیشتر است.

$$f(n) = O(n^{1-\epsilon}) \Rightarrow T(n) = \Theta(n^1)$$

مثال ۲: $T(n) = T\left(\frac{n}{2}\right) + 1$

داریم، $f(n) = \Theta(\log_r n)$ و $g(n) = n^{\log_2 2} = n^1 = 1$.

مثال ۳: $T(n) = 3T\left(\frac{n}{3}\right) + n \log_r n$

چون، $f(n) = \Theta(n \log_r n)$ ، داریم $f(n) = \Omega(n^{1/2+1/3+\dots/3})$ و $g(n) = n^{\log_3 3} = n^{1/3} = 1$.

مثال ۴: $T(n) = 7T\left(\frac{n}{7}\right) + n \log_r n$

برای این مثال، $f(n) = n \lg_r n = n^{\log_r 7} = n$ و $g(n) = n^{\log_7 7} = n^1 = n$. ولی اختلاف به صورت نمایی نیست یعنی هیچ‌یا نمی‌توان پیدا کرد که برای کلیه‌ی n ‌های بزرگ، رابطه‌ی $n \log_r n < n^1$ برقرار باشد. بنابراین این مسئله را باید از روش دیگری، مثلاً استقرای حل کرد. جواب $T(n) = O(n \log_r n)$ است که با استقرای اثبات می‌شود.

۲.۱ روش‌های حل رابطه‌های بازگشتی

۴.۷.۱ رابطه‌های بازگشتی همگن^۱

هدف این مقاله معرفی و بررسی رابطه‌های بازگشتی همگن و ارایهٔ روش‌های حل آن‌هاست. به مثال‌های زیر توجه کنید:

مثال ۱. به چند طریق می‌توان صفحه‌ای $n \times 2$ را با موزاییک‌های 1×2 فرش کرد؟
 حل: اگر جدول $n \times 2$ را بتوان با k روش مختلف با موزاییک‌های 1×2 فرش کرد به راحتی می‌توان ثابت کرد که $f_n = f_{n-1} + f_{n-2}$. به این ترتیب که اگر در متون $n-1$ جدول یک موزاییک 1×2 به صورت عمودی گذاشته شود، جدولی $(n-1) \times 2$ می‌ماند که باید با موزاییک‌های 1×2 فرش شود که به f_{n-1} طریق ممکن است. در غیر این صورت دو متون $n-1$ و n با دو موزاییک 1×2 افقی بوشانده شده‌اند، که جدولی $(n-2) \times 2$ می‌ماند که باید با موزاییک‌های 1×2 فرش شود که به f_{n-2} طریق ممکن است. پس در کل تعداد روش‌های فرش کردن جدول $n \times 2$ با موزاییک‌های 1×2 $f_{n-1} + f_{n-2}$ است، یعنی $f_n = f_{n-1} + f_{n-2}$. این رابطه، دنباله‌ی معروفی را مشخص می‌کند که با مقادیر اولیهٔ $f_1 = 1$ و $f_2 = 1$ دنباله‌ی فیبوناچی نام دارد.

تمرین ۱ به چند طریق می‌توان صفحه‌ای $n \times 3$ را با موزاییک‌های 1×2 فرش کرد؟ (رابطه‌ای بازگشتی را باید و آن را حل کنید).

دقت می‌کیم که رابطهٔ $f_n = f_{n-1} + f_{n-2}$ هر عضو دنباله را بر حسب اعضای قبلی دنباله بیان می‌کند، مثلاً برای پیدا کردن f_2 می‌توان از رابطهٔ $f_2 = f_1 + f_0$ استفاده کرد و بدست آورد: $f_2 = 0 + 1 = 1$. حال به همین ترتیب با جمع f_2 و f_1 می‌توان f_3 را به دست آورد. ضمناً برای ساختن دنباله باید دو عضو اول آن را داشته باشیم تا با استفاده از آنها f_2 و f_3 و همین‌طور تا f_n مشخص شوند. حل رابطه‌ی بازگشتی به این معناست که ما هر عضو دنباله‌ی f_n را بدون استفاده از اعضای قبلی و مستقل از آن اعضا بیان کنیم.
 اگر هم‌اعدداهای حقیقی باشند، به رابطه‌ی بازگشتی زیر رابطه‌ی بازگشتی همگن از درجهٔ k می‌گویند:

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \dots + c_k a_{n-k} \quad (1)$$

در این رابطه، a_n بر حسب k جمله‌ی قبیل آن داده شده است و برای ساخت دنباله به k جمله‌ی اول آن احتیاج داریم. تابع $(n)g$ را یک جواب دنباله‌ی بازگشتی فوق می‌نامند. اگر دنباله‌ی $(n)g(n)$ در رابطه‌ی بازگشتی صدقی کند.

قضیه ۱. اصل برهم نهی: اگر $(n)g_i(n)$ (یعنی برای $i = 1, \dots, r$)

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \dots + c_k a_{n-k} + f_i(n) \quad (2)$$

باشند، آنگاه هر ترکیب خطی از این k جواب به صورت $(n)g_r(n) + A_1 g_1(n) + A_2 g_2(n) + \dots + A_r g_r(n)$ دارد. که در آن A_i ها اعدادی حقیقی‌اند، پاسخی برای رابطه‌ی بازگشتی شماره‌ی i است. بدويژه، جون در رابطه‌های بازگشتی همگن $= f_i(n)$ ، هر ترکیب خطی جواب‌های یک رابطه‌ی بازگشتی همگن باز یک جواب همان رابطه‌ی بازگشتی است.

اثبات: فرض کنیم، $(n)h(n) = A_1 g_1(n) + A_2 g_2(n) + \dots + A_r g_r(n)$ چون $(n)g_i(n)$ یک جواب $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \dots + c_k a_{n-k} + f_i(n)$

$$g_i(n) = c_1 g_i(n-1) + c_2 g_i(n-2) + \dots + c_k g_i(n-k) + f_i(n)$$

این سخن نویسط آنای وعاب سیر رکنی داشت جوی ساین دانستند، نهیه شده است. با شکر اینسان.

پس نتیجه می‌شود:

$$h(n) = c_0 h(n-1) + c_1 h(n-2) + \cdots + c_k h(n-k) + A_1 f_1(n) + \cdots + A_r f_r(n)$$

بنابراین حکم قضیه ثابت است.

حل رابطه‌های بازگشتی همگن

رابطه‌های بازگشتی همگن حل ساده‌ای دارند که بدین صورت است: اگر $x = g(n)$, جواب رابطه‌ی بازگشتی همگن شماره‌ی ۱ باشد، داریم:

$$x^n - c_1 x^{n-1} - c_2 x^{n-2} - \cdots - c_k x^{n-k} = 0$$

با به عبارت دیگر،

$$x^k - c_1 x^{k-1} - \cdots - c_k = 0 \quad (3)$$

است، یعنی x جواب معادله درجه k فوق است. این معادله را معادله‌ی سرشتمای معادله‌ی متشکله^{۱۰} رابطه‌ی بازگشتی می‌نامیم. اگر x ، ریشه‌ی معادله‌ی سرشتمای باشد، بدینهی است که $x^n = a_n$ یک جواب رابطه‌ی بازگشتی است و با بر قضیه‌ی قبلي هر ترکیب خطی از x ‌ها هم یک جواب رابطه‌ی بازگشتی است. ضمناً این جواب‌ها پایه‌ای برای مجموعه جواب این رابطه می‌باشند. البته چگونگی تحقیق این مطلب به مقدماتی از جبر خطی احتیاج دارد. با اثبات این مطلب می‌توان گفت: تمام جواب‌های رابطه‌ی ۱ به صورت ترکیبی خطی از x ‌ها است. (چگونگی اثبات را تحقیق کنید). به طور مثال ترکیب خطی:

$$a_n = t_1 x_1^n + t_2 x_2^n + \cdots + t_k x_k^n; \quad (4)$$

را در نظر می‌گیریم. اگر در این رابطه‌ی بازگشتی k عنصر اول این دنباله داده شده باشد، باید k معادله‌ی زیر برقرار باشد:

$$\begin{cases} a_0 &= t_1 + t_2 + \cdots + t_k \\ a_1 &= t_1 x_1 + t_2 x_2 + \cdots + t_k x_k \\ \vdots &\vdots \quad \vdots \quad \vdots \quad \vdots \\ a_{k-1} &= t_1 x_1^{k-1} + t_2 x_2^{k-1} + \cdots + t_k x_k^{k-1} \end{cases}$$

این یک دستگاه k معادله و k مجهول می‌باشد. (محبول‌ها t_1, t_2, \dots, t_k هستند). با توجه به تشکیل دترمینان ضرائب، این نتیجه به دست می‌آید که اگر x ‌ها متعابر باشند این دستگاه معادلات یک جواب منحصر به فرد دارد، یعنی بادادن k عصر اول دنباله، می‌توان جوابی منحصر به فرد برای دنباله پیدا کرد. (تحقیق کنید چرا در صورت متعابر بودن x ‌ها، این دستگاه معادلات یک جواب منحصر به فرد دارد! برای این کار باید دترمینان ماتریس ضرائب را تشکیل دهیم، می‌توانید به مرجع [?] مراجعه کنید).

مثال ۲. دستاله‌ی فیبوناچی که در مثال (۱) تعریف شده است، را حل کنید.

حل: معادله‌ی سرشتمای این رابطه به صورت $x = 1 - x - x^2$ است از لرسته‌های آن $x^2 - x - 1 = 0$ و

$x_1 = \frac{1 + \sqrt{5}}{2}$ است. پس

characteristic equation^{۱۱}

$$f_n = t_1 \left(\frac{1+\sqrt{5}}{2} \right)^n + t_2 \left(\frac{1-\sqrt{5}}{2} \right)^n$$

و با توجه به مقادیر اولیه داریم:

$$\begin{cases} t_1 + t_2 = f_0 = 0 \\ t_1 \left(\frac{1+\sqrt{5}}{2} \right) + t_2 \left(\frac{1-\sqrt{5}}{2} \right) = f_1 = 1 \end{cases}$$

و از این معادله‌ها نتیجه می‌شود:

$$t_1 = \frac{1}{\sqrt{5}}, t_2 = -\frac{1}{\sqrt{5}}$$

$$f_n = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right) \quad (5)$$

قابل توجه است، جمله‌ی x^{n-1} بازگشتن n بسیار کوچک می‌شود و با توجه به اینکه f_n عددی حسابی است، اگر x را نزدیکترین عدد صحیح به x تعریف کنیم داریم:

$$\langle x \rangle = x + \frac{1}{2}$$

و با این تعریف:

$$f_n = \left\langle \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^n \right\rangle \quad (6)$$

در باقی جوابهای دستگاه به دست آمده برای باقی ضرایب، این شرط را قرار دادیم که، ها متمایز باشند. حال اگر ریشه‌ی مضاعف درجه‌ی ۲ معادله‌ی سرشتمان باشد، به راحتی قابل تحقیق است که $a_n = nx^n$ نیز یک جواب رابطه‌ی بازگشتی است (ایات با مشتق‌گیری از معادله‌ی سرشتمان است، به این وسیله که ریشه‌ی مضاعف، ریشه‌ی مشتق معادله‌ی سرشتمان است). به همین طریق می‌توان استدلال کرد که اگر x ریشه‌ی مضاعف درجه ۳ باشد، n^2x^n نیز یک جواب رابطه‌ی بازگشتی است. در حالت کلی اگر x ریشه‌ی مضاعف درجه p باشد،

$$g(n) = t_0 x^n + t_1 nx^n + t_2 n^2 x^n + \dots + t_{p-1} n^{p-1} x^n$$

جوابی برای رابطه‌ی بازگشتی است.

مثال ۲. رابطه‌ی بازگشتی زیر را حل کنید:

$$a_n = -a_{n-1} + 3a_{n-2} + 5a_{n-3} + 2a_{n-4} \quad (n \geq 5)$$

حل: ریشه‌های معادله سرشتمانی آن به صورت زیر درآید:

$$x^4 + x^3 - 2x^2 - 5x - 2 = 0 \Rightarrow a_n = t_1 (-1)^n + t_2 n^2 (-1)^n + t_3 2^n$$

و با معلوم بودن $a_1 = 0$, $a_2 = 0$, $a_3 = 0$ به دست می‌آید. مثلاً در حالت $t_1 = 4$ و $a_1 = -3$ و $a_2 = 22$ و $a_3 = 0$ و $a_4 = -7$

از دستگاههای متناظر جوابهای $t_1 = 1$, $t_2 = -1$, $t_3 = 2$ و $t_4 = -1$ به دست می‌آید.

مثال بعد، حالتی را بررسی می‌کند که در آن معادله‌ی سرشتمان ریشه‌ی حقیقی ندارد.

مثال ۴. رابطه‌ی بازگشتی $a_n = \alpha a_{n-1} + \beta a_{n-2}$ را حل کنید.
حل: معادله‌ی سرشت‌نمای آن به صورت $x^2 - \alpha x - \beta = 0$ است که جواب غیر حقیقی مقابل را دارد.
 $\alpha = 1 - i, \beta = 1 + i$

$$\begin{cases} \alpha = \sqrt{2} (\cos(\frac{\pi}{4}) + i \sin(\frac{\pi}{4})) \\ \beta = \sqrt{2} (\cos(\frac{\pi}{4}) - i \sin(\frac{\pi}{4})) \end{cases}$$

$$\Rightarrow a_n = A\alpha^n + B\beta^n = \left\{ (\sqrt{2})^n \left[A \left(\cos\left(\frac{n\pi}{4}\right) + i \sin\left(\frac{n\pi}{4}\right) \right) + B \left(\cos\left(\frac{n\pi}{4}\right) - i \sin\left(\frac{n\pi}{4}\right) \right) \right] \right\}$$

$$\Rightarrow a_n = \sqrt{2} \left(C \cos\left(\frac{n\pi}{4}\right) + D \sin\left(\frac{n\pi}{4}\right) \right)$$

$$a_0 = 1, a_1 = 0 \Rightarrow C = 1, D = -1$$

$$\Rightarrow a_n = \sqrt{2} \left(\cos\left(\frac{n\pi}{4}\right) - \sin\left(\frac{n\pi}{4}\right) \right)$$

لازم به نذکر است در این معادلات از رابطه‌ی مهم زیر استفاده شده است که با استفاده از استقرا ثابت می‌شود.

$$(\cos \theta + i \sin \theta)^n = \cos(n\theta) + i \sin(n\theta)$$

۵.۷.۱ رابطه‌ها بازگشتی غیر همگن با ضرایب ثابت:

رابطه‌ی بازگشتی درجه‌ی k زیر غیر همگن است:

$$a_n = c_1 a_{n-1} + \cdots + c_k a_{n-k} + f(n)$$

اگر A_n در رابطه‌ی بازگشتی همگن ۱ صدق کند و B_n در رابطه‌ی بازگشتی غیر همگن صدق کند آنگاه $A_n + B_n$ تجزیه در رابطه‌ی غیر همگن صدق می‌کند. بر علاوه:

$$c_1(A_{n-1} + B_{n-1}) + \cdots + c_k(A_{n-k} + B_{n-k}) + f(n) =$$

$$(c_1 A_{n-1} + \cdots + c_k A_{n-k}) + (c_1 B_{n-1} + \cdots + c_k B_{n-k} + f(n)) = A_n + B_n$$

در این حالت A_n را جواب قسمت همگن رابطه و B_n را یک جواب خاص آن گویند. مثلًا اگر $B_n = pn^r + qn + r$. $f_n = \tau - \tau n^r$. $A_n = t_1 \tau^n$. $a_n = \tau a_{n-1} + \tau - \tau n^r$ باشد، آنگاه $B_n = pn^r + qn + r = \tau(p(n-1)^r + q(n-1) + r) + \tau - \tau n^r$

در نتیجه، برای این که این رابطه یک اتحاد برای n باشد. داریم: $D = \tau$, $C = \tau$, $B = 1$, $a_n = A_n + B_n = t_1 \tau^n + n^r + \tau$ و خواهیم داشت:

$$a_n = A_n + B_n = t_1 \tau^n + n^r + \tau$$

۱. روش‌های حل رابطه‌های بازگشتی

با فرض $a_0 = n$ داریم: $a_1 = 1$ پس:

$$a_n = 2^n + n^r + 2n + 2$$

۶.۷.۱ حل رابطه‌ها بازگشتی غیر همگن:

اگر رابطه‌ی بازگشتی غیر همگن را به صورت زیر داشته باشیم:

$$(7) \quad a_n = c_1 a_{n-1} + c_2 a_{n-2} + \dots + c_k a_{n-k} + b^n P(n)$$

که در آن b ثابت است و $P(n)$ چند جمله‌ای، از درجه‌ی d بر حسب n باشد، برای این رابطه، مشابه رابطه‌ها بازگشتی همگن معادله‌ی سرشتمان به صورت زیر تعریف می‌شود:

$$(8) \quad (x^k - c_1 x^{k-1} - c_2 x^{k-2} - \dots - c_k)(x - b)^{d+1} = 0$$

با داشتن این معادله و به دست آوردن ریشه‌های (x_i) آن مشابه قبل، جوابهای این معادله به صورت ترکیب خطی $\sum c_i x_i^n$ می‌شود. (تحقیق این موضوع به عهده‌ی شما). به طور کلی ریشه‌های معادله‌ی سرشتمانی یک رابطه‌ی بازگشتی مشخص کننده‌ی جوابهای رابطه‌ی بازگشتی است. با داشتن این معادله‌ی سرشتمانی برای رابطه‌ها بازگشتی بسیاری از این رابطه‌ها با روشن مشابه رابطه‌ها بازگشتی همگن، به راحتی قابل حل است.

مثال ۵.

* (الف) رابطه‌ی بازگشتی $2^n = 2a_{n-1} + (n+5)$ را حل کند.

* (ب) رابطه‌ی بازگشتی $a_n = 2a_{n-1} + n$ را حل کند.

حل:

* (الف) معادله‌ی سرشتمانی این رابطه به صورت $= 2(x - 2)(x - 2 - 1)$ در می‌آید، پس:

$$a_n = t_1 2^n + t_2 2^n + t_3 n 2^n$$

که با توجه به حالت‌های اولیه داده شده، میتوان t_1, t_2 را به دست آورد.

* (ب) معادله‌ی سرشتمانی آن $= 2(x - 2)(x - 2 - 1)$ است، پس $a_n = t_1 2^n + t_2 + t_3 n$ است.

در حالت کلی معادله‌ی سرشتمانی رابطه‌ی بازگشتی مقابل

$$(9) \quad a_n = c_1 a_{n-1} + c_2 a_{n-2} + \dots + c_k a_{n-k} + b_1^n P_1(n) + b_2^n P_2(n) + \dots$$

به صورت،

$$(10) \quad (x^k - c_1 x^{k-1} - c_2 x^{k-2} - \dots - c_k)(x - b_1)^{d_1+1}(x - b_2)^{d_2+1} \dots = 0$$

است.

مثال ۶. رابطه‌ی بازگشتی $a_n = 2a_{n-1} + n + 2^n$ را حل کند.

فصل ۱ روش‌های تحلیل الگوریتم‌ها

۲۴

حل: معادله سرشت نمای آن به صورت $a_n = (x - 2)(x - 2)(x - 2)$ است. پس

$$a_n = t_1 + t_2 n + t_3 n^2 + t_4 n^3$$

$$a_n = -2 - n + 2n^2 + n^3 \quad \text{داریم:}$$

مدى ترتیب سیاری از رابطه‌ها بازگشتی با ضرایب ثابت حل می‌شوند.

حال با حل مسئله ای کاربردی به روشی دیگر در حل رابطه‌ها بازگشتی توجه می‌کیم:

مثال ۷. مسئله (بیست و پنجمین المپیاد جهانی ریاضی): A و E را دو رأس رویه‌روی یک Δ ضلعی منتظم می‌گیریم، قورباغه‌ای از رأس A . آغاز به جهیدن می‌کند و هر بار به رأس مجاور می‌پرد. ولی وقتی به رأس E رسید، همانجا متوقف می‌شود. a_n را تعداد مسیرهایی می‌گیریم که قورباغه با n جهش از طریق آنها از A به E برسد. ثابت کنید:

$$a_{2n} = \frac{1}{\sqrt{2}}(x^{n-1} - y^{n-1})$$

$$\text{که در آن } \sqrt{2} + x = 2 \text{ و } \sqrt{2} - y = 2$$

اثبات: Δ ضلعی رویه‌روی را در نظر می‌گیریم، اگر b_n تعداد مسیرهایی باشد که قورباغه از آنها با n جهش از A به E برسد. اگر قورباغه بخواهد از A به E برسد، در دو جهش اول پا به C می‌رسد پا به G می‌رسد یا پا به A بررسی گردد که به دو طریق می‌تواند به A بازگردد $[ABA, AHA]$. حال از جایی که الان به آن به E رسیده باید شروع کند و با $2 - n$ جهش به E برسد، بنابراین داریم: $a_n = 2b_{n-2} + ta_{n-2}$. در مورد b_n نیز مشابه می‌توان گفت:

$$(جرا؟) b_n = 2b_{n-2} + a_{n-2}$$

نخست دقت می‌کنیم چون بین A و E . Δ ضلع وجود دارد، و جهش‌های رفت و برگشت با هم تعداد زوجی می‌سازند، برای رفتن از A به E حتماً تعداد زوجی حرکت لازم است. پس $a_{2n-1} = 0$. برای حالات‌های زوج دو رابطه‌ی بازگشتی $a_{2n} = 2b_{n-2} + ta_{n-2}$ و $a_n = 2b_{n-2} + ta_{n-2}$ داریم. حال دو روش وجود دارد، روش اول: از تفاضل دو رابطه بدست می‌آید: $b_{n-2} = a_{n-2} - a_{n-4}$. حال اگر $a_{n-2} = a_{n-4} = c_n$ در نظر بگیریم، رابطه‌ی همگن $(n > 2)c_n = 2c_{n-2} - 2c_{n-4}$ داریم: می‌آید که با توجه به حالات‌ای اولیه $c_0 = 0$ و $c_1 = 1$ خواهیم داشت:

$$a_{2n} = c_n = \frac{1}{\sqrt{2}}((2 + \sqrt{2})^{n-1} - (2 - \sqrt{2})^{n-1})$$

(ما تشکیل معادله‌ی سرشت نمای و حل آن بدست می‌آید) حال مشابه مثال ۵. به دلیل اینکه $1 < \sqrt{2} - 2$. می‌توان تحقیق کرد:

$$f_n = \left(\frac{(2 + \sqrt{2})^{n-1}}{\sqrt{2}} \right) \quad (11)$$

روش دوم: این روش با آنچه تا به حال گفته‌یم متفاوت است. ما به این روش در حل این مسئله بسته می‌کنیم:

$$\begin{aligned} a_n &= 2a_{n-2} + 2b_{n-2} \\ b_n &= a_{n-2} + 2b_{n-2} \end{aligned}$$

حال اگر بردار \vec{v}_m را به صورت $\vec{v}_m = \begin{pmatrix} a_m \\ b_m \end{pmatrix}$ تعریف کنیم، باید داشته باشیم: $\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \vec{v}_m = \vec{v}_{m-2}$

$$\vec{v}_m = \begin{pmatrix} * \\ * \end{pmatrix}$$

۲۴

۱. روش‌های حل رابطه‌های بازگشتهای

برای تعیین بردار v_m با این حالت خاصیت مقادیر ویژه‌ی ماتریس T را تعیین می‌کنیم: این مقادیر ریشه‌های معادله‌ی مفسر ماتریس می‌باشند:

$$\begin{vmatrix} 2 - \lambda & 2 \\ 1 & 2 - \lambda \end{vmatrix} = \lambda^2 - 4\lambda + 2 = 0$$

و بنابراین: $\lambda_1 = 2 + \sqrt{2}$ و $\lambda_2 = 2 - \sqrt{2}$. بردارهای ویژه‌ی ماتریس T ، یعنی u_1 و u_2 دارای این ویژگی هستند که $T[Tu_i] = \lambda_i u_i$ و $T^m u_i = \lambda_i^m u_i$ می‌توان آنها را پیدا کرد:

$$u_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ \sqrt{2} \end{pmatrix}, \quad u_2 = \frac{1}{\sqrt{2}} \begin{pmatrix} -1 \\ \sqrt{2} \end{pmatrix}$$

بنابراین v_m یک ترکیب خطی از u_1 و u_2 است. یعنی:

$$v_m = \lambda_1 u_1 + \lambda_2 u_2 \Rightarrow v_m = T^{(m-1)} v_1 = \lambda_1^{m-1} u_1 + \lambda_2^{m-1} u_2 = \begin{pmatrix} a_{1m} \\ b_{1m} \end{pmatrix}$$

و بدین ترتیب،

$$\Rightarrow a_{1m} = \frac{1}{\sqrt{2}} [(2 + \sqrt{2})^{m-1} + (2 - \sqrt{2})^{m-1}] \quad (12)$$

تمرین‌ها

۱. روی محیط دایره‌ای چند کارت کوچک سیاه و سفید قرار داده‌ایم. دو نفر به نوبت این عمل را انجام می‌دهند، اولی همه‌ی کارت‌های سیاهی که در مجاورت یک سفید باشند، بر می‌دارد و دومی روی کارت‌های سفید مجاور سیاه همین کار را انجام می‌دهد. اگر ۴۰ کارت وجود داشته باشد، آیا ممکن است پس از انجام ۲ حرکت فقط یک کارت باقی بماند؟ اگر کارت‌ها ۱۰۰۰ نا باشند، حداقل چند حرکت لازم است؟ اگر ۷ کارت دور دایره باشند، حداقل چند حرکت لازم است؟

۲. تعداد زیردرخت‌های فراگیر گراف‌های زیر چند ناست؟

- تردبان "نایی" که از دو مسیر "نایی" که با یک تطابق به هم وصل شده‌اند، تشکیل شده است ($A_4 \times A_4$)
- دور "نایی" که از یک دور "نایی" و یک رأس که به همه‌ی رؤوس دور متصل است، تشکیل شده است.

^{۱۱} زیردرخت فراگیر یک گراف زیرگراف هستدی از گراف است که دور نداشته باشد و شامل همه‌ی رؤوس باشد.

فصل ۲

مرتب‌سازی و مرتبه‌ی آماری

There is nothing more difficult to take in hand, more peritous to conduct, or more uncertain in its success, than to take the lead in the introduction a new order of things. (Niccolo Machiavelli, The prince, 1513)

۱.۲ الگوریتم‌های مرتب‌ساز

مساله‌ی مرتب‌سازی: n عنصر با کلیدهای a_1, a_2, \dots, a_n و رابطه‌ی ترتیب کامل^۱ که داده شده‌اند جایی قرار گیرد که از عناصر داده شده را پیدا کنید به طوری که:

$$a_{1,1} \leq a_{2,1} \leq \dots \leq a_{n,1}$$

الگوریتم‌های مرتب‌ساز^۲ را از سه جنبه می‌توان بررسی کرد:

۱. از نظر موقعیت داده‌ها در زمان مرتب کردن

- الگوریتم‌های مرتب‌ساز داخلی^۳: که در آن‌ها همه‌ی داده‌ها هنگام مرتب شدن در حافظه هستند. بنابراین دست‌یابی به داده‌ها سریع صورت می‌گیرد.
- الگوریتم‌های مرتب‌ساز خارجی^۴: که در آن‌ها همه‌ی داده‌ها هنگام مرتب شدن به طور همزمان در حافظه نیستند. بنابراین دست‌یابی به آن‌ها کند است. در این الگوریتم‌ها تعداد دست‌یابی به حافظه‌ی جانشینی، به دلیل کندی این عمل، اهمیت زیادی دارد بخصوص اگر داده‌ها نرتقی^۵ باشند.

۲. از نظر حفظ ترتیب نسبی عناصر

مجموعه‌ی A رابطه‌ی ترتیب مجزی^۶ (partial order)^۷ است اگر عرب‌ساز، بارگذاری و متعدد باشد. رابطه‌ی ترتیب حریق^۸ (total order)^۹ کامل است. اگر مراوی هر $A, B \in A$ داشته باشیم $R_B \subseteq R_A$ باشد. الگوریتم‌های مرتب‌سازی^{۱۰} sorting algorithms^{۱۱} internal sorting algorithms^{۱۲} external sorting algorithms^{۱۳} sequential^{۱۴}

- الگوریتم‌های متعادل^۶، که در آن‌ها ترتیب نسبی عناصر با کلیدهای یکسان قبل و بعد از عمل مرتبسازی پکی باشد.
 - الگوریتم‌های غیرمتعادل^۷، که در آن‌ها ترتیب نسبی عناصر مساوی در انتهای لزوماً حفظ نشود.
۳. از نظر نحوه مرتب کردن داده‌ها
- الگوریتم‌های مبتنی بر مقایسه^۸، که بر اساس مقایسه کلیدهای عناصر عمل می‌کنند. این الگوریتم‌ها اطلاعات دیگری از داده‌ی ورودی ندارند و فقط با انجام مقایسه بین کلیدهای عناصر می‌توانند ترتیب نسبی آن‌ها را پیدا کنند.
 - الگوریتم‌هایی که بر اساس مقایسه عناصر عمل نمی‌کنند^۹. این الگوریتم‌ها با استفاده از اطلاعاتی که از قبیل درباره‌ی نوع کلیدها موجود است، و در شرایط خاص، از روش‌هایی استفاده می‌کنند که در آن‌ها کلیدهای عناصر با هم مقایسه نمی‌شوند. بدینهی است که حد پایین برای هر الگوریتم مرتب‌ساز، از جمله الگوریتم‌هایی که مبتنی بر مقایسه نیستند $\Omega(n)$ است. برای الگوریتم‌های مبتنی بر مقایسه حد پایین هم در بدترین حالت و حالت متوسط $\Omega(n \log n)$ است.

ابندا به بررسی الگوریتم‌هایی که بر اساس مقایسه عناصر عمل نمی‌کنند می‌پردازیم که از آن‌ها Count Sort، Bucket Sort و Radix Sort می‌باشد و هر مولفه حوزه‌ی محدود و قابل شمارش دارد. قبل از ارایه‌ی الگوریتم، به یک تعریف توجه کنید:

$$\begin{aligned}\vec{a} &= a_1, a_2, \dots, a_i, \dots, a_n \\ \vec{b} &= b_1, b_2, \dots, b_i, \dots, b_m\end{aligned}$$

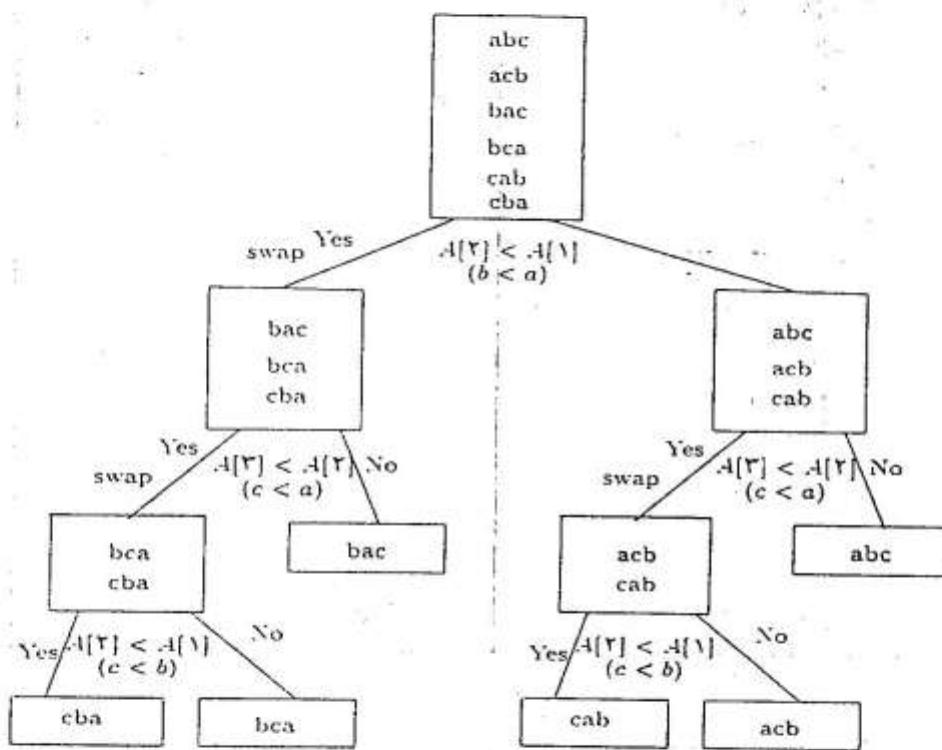
می‌گوییم $b_i \leq a_j$ اگر برای $n \leq i < k$ داشته باشیم، $b_i = b$ و $a_j = a$ و $b_i < a_j$ یا $m < n$ و برای $n \leq i \leq k$ داشته باشیم $b_i = a_i$. عمل مرتبسازی در مواردی که کلیدهای رشته باشند بر اساس ترتیب الفبایی است.

۱.۱.۲ درخت تصمیم

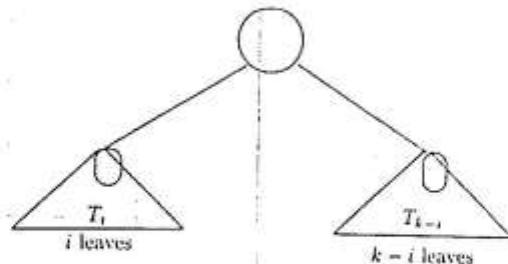
یک الگوریتم که بر روی عناصر ورودی عمل مقایسه انجام می‌دهد را می‌توان با «درخت تصمیم^{۱۰}» مدل کرد. به طور مثال، الگوریتم مرتب‌سازی «مبتنی بر درج» یا Insertion Sort روی مه عنصر a_1, a_2, \dots, a_n به صورت شکل ۱.۲ مدل شده است. بدینهی است که هر درخت تصمیم یک الگوریتم مرتب‌ساز یک درخت دودویی کامل است که برگ‌های آن جای‌گشتهای مختلف ورودی هستند.

قضیه ۲. یک درخت دودویی با ارتفاع $\lceil \lg n \rceil$ برگ دارد.

stable ^{۱۱}
unstable ^{۱۲}
comparison sort ^{۱۳}
non-comparison sort ^{۱۴}
Lexicographic Ordering ^{۱۵}
decision tree ^{۱۶}



شکل ۱.۲: درخت ترتیب



شکل ۲.۲: اثبات قضیه

این قضیه با استفرا روی n اثبات می‌شود (امتحان کنید).

قضیه ۳. ارتفاع یک درخت تصمیم که n عنصر را مرتب می‌کند حداقل $\lceil \log n! \rceil$ است.

□ اثبات. این درخت تصمیم حداقل $n!$ برگ دارد، بنابراین ارتفاع حداقل $\lceil \log n! \rceil$ است.

نتیجه: هر الگوریتم مرتب‌ساز متنی بر مقایسه، برای مرتب کردن n عنصر $\lceil \log n! \rceil$ بدترین حالت حداقل

مقایسه انجام می‌دهد.

می‌دانیم که $n! \leq n^n$ پس $n \lg n \leq \lceil \log n! \rceil$. ولی این حد بالای ضعیفی است. تقریب استرلینگ^{۱۲} حد بدتری را بدست می‌دهد. براساس این تقریب داریم:

$$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \left(1 + O\left(\frac{1}{n}\right)\right)$$

با استفاده از این فرمول می‌توان اثبات کرد که:

$$n! = \Theta(n^n)$$

$$n! = \Theta(2^n)$$

$$\lg(n!) = \Theta(n \lg n)$$

قضیه ۴. اگر کلبه‌ی جای گشت‌های یک ترتیب T ای ب احتمال بکسان در ورودی T ظاهر شوند. آن گاه متوسط عمق برگ‌های این درخت حداقل $\lceil \log n! \rceil$ خواهد بود.

اثبات. فرض می‌کنیم $D(T)$ مجموع عمق برگ‌های یک درخت دوری T و $D(m)$ کوچک‌ترین مقدار $D(T)$ برای کلبه‌ی درخت‌های دوری m باشد. با استفرا ثابت می‌کنیم که

$$1. \text{ پایه: } m = 1 \text{ واضح است.}$$

¹² Stirling's Approximation

۲. فرض: برای $k < m$ درست است.

۳. حکم: T با k بزرگ را در نظر بگیرید (شکل ۲.۲).

$$\begin{aligned} D(T) &= i + D(T_i) + (k - i) + D(T_{k-i}) \\ D(k) &= \min_{1 \leq i \leq k} \{k + D(i) + D(k - i)\} \\ D(k) &\geq k + \min_{1 \leq i \leq k} \{i \log i + (k - i) \log (k - i)\} \end{aligned}$$

با استقرار ثابت می‌شود که برای اعداد طبیعی مقدار کمینه در $\frac{k}{2} = i$ اتفاق می‌افتد. پس $k \geq i \log i$

$$D(m) \geq m \log m$$

□

۲.۲ الگوریتم‌های مرتب‌ساز خطی

۱.۲.۲ Count Sort

ورودی n عنصر با کلیدهای بین ۱ تا m می‌باشد. می‌خواهیم این عناصر را حابه‌جا بکنیم به طوری که بر اساس کلیدهایشان مرتب شوند. در این الگوریتم ورودی در آرایه‌ی A قرار دارد و خروجی در آرایه‌ی B ذخیره می‌شود. الگوریتم ابتدا تعداد عناصری را که کلیدشان برابر یک مقدار کلید، مثلاً r است معین می‌کند و از روی آن بزرگترین اندیس در آرایه‌ی B برای عناصر با کلید r را محاسبه می‌کند. در انتهایا یک بار بررسی آرایه‌ی A . هر عنصر آن در جای درست در آرایه‌ی B قرار می‌گیرد.

```
Count_Sort(A,B,m)
1   for i<-1 to m
2     do c[i] <- 0
3   for i <-1 to length(A)
4     do C[A[i]] <- C[A[i]]+1
5   for i <-2 to m
6     do C[i] <- C[i]+C[i-1]
7   for i <- length(A) downto 1
8     do B[C[A[i]]] <- A[i]
9     C[A[i]] <- C[A[i]]-1
```

یک عنصر در جای متناسب قرار می‌گیرد و چون در هر مرحله عناصر قابلی که در جای خود قرار گرفته‌اند، دست‌کاری نمی‌شوند بعد از n مرحله همه‌ی عناصر مرتب می‌شوند. واضح است که زمان اجرا $O(n)$ می‌باشد. مقدار $\{i\}$ از آرایه $C[1..m]$ در انتهای گام چهارم تعداد عناصر با کلید i را نشان می‌دهد و در انتهای گام ششم این درایه تعدیله عناصری که کلید آن‌ها حداقل i است را ذخیره می‌کند. در هر بار حلقه‌ی آخر، یک عنصر از

فصل ۲ مرتبسازی و مرتبه‌ی آماری

۲۲

در جای نهایی خود در آرایه A قرار می‌گیرد. دلیل آن که این حلقه از انها به ابتدای A نکرار می‌شود آن است که الگوریتم متعادل شود. زمان اجرای کل الگوریتم $O(n+m)$ می‌باشد که اگر $m > n$ باشد زمان اجرای کل $O(n)$ خواهد بود.

اگر کلیدهای عناصر اعداد A نا-باشند (از هر کلید یک عدد)، می‌توان با الگوریتم زیر آن‌ها را به صورت «درجا» (بدون استفاده از آرایه کمکی) مرتب کرد.

عملیات درجا برای مرتبسازی متعادل

[۷.۶]

```
Count_Sort(A,n)
1   for i<-1 to n
2     do while A[i].key <> i
3       do Swap(A[i],A[A[i].key])
```

در این الگوریتم با هر بار تغییض، حداقل یک عنصر در جای نهایی خودش قرار می‌گیرد (چرا؟) هم‌چنین اگر از هر کلید بیش از یک عدد داشته باشیم الگوریتم ممکن است در حلقه بیفتد.

۲.۲.۲ الگوریتم Radix Sort

این یک الگوریتم قدیمی است که در مانیعنی‌های مرتب‌ساز کارتهای از آن استفاده می‌شود. در این الگوریتم (اگر برای مرتب کردن اعداد به کار گرفته شود) ورودی‌ها را ابتدا براساس کم‌ارزش‌ترین رقم و سپس دویمن کم‌ارزش‌ترین رقم و به همین ترتیب مرتب می‌کنیم. برای این کار باید از یک الگوریتم مرتب‌ساز متعادل مانند Count_Sort استفاده کرد.

[۷.۷] از این مرتبسازی سریع تر

```
Radix_Sort(A, d)
1   for i<-1 to d
2     do sort array A on digit i by a stable sort
```

ا) تعداد رقم‌های ورودی می‌باشد. اگر تعداد رقم‌های ورودی بکسان باشند با گذاشتن صفر در سمت چپ آن‌ها تعداد رقم‌ها را ساوهی می‌کنیم.

درستی الگوریتم را می‌توان با استقرار بررسی اثبات نمود. به این صورت که در انتهای مرحله‌ی i ام، عناصر بر حسب i رقم آخرین مرتب می‌شوند. پایه‌ی استقرار برای $=0$ روش است. اگر این امر برای $i = k$ درست باشد، با توجه به این که از یک الگوریتم متعادل استفاده می‌کنیم، در انتهای مرحله‌ی $i+1$ ام نیز ادعا درست است.

زمان اجرای Radix Sort به الگوریتم ثانویه به کار رفته نیز بستگی دارد. اگر از Count Sort استفاده کنیم زمان اجرا از $O(dn)$ خواهد بود.

Radix Sort را می‌توان تعمیم داد برای حالی که ورودی آرایه‌ای از رکوردها باشد که هر رکورد دارای کلیدی با k مؤلفه به نام‌های f_1, \dots, f_k از داده‌گونه‌های به ترتیب $(1, \dots, k)$ باشد.

۳۲

۲.۲.۲ الگوریتم Bucket Sort

در این الگوریتم ورودی می تواند لیست A با n رکورد، که هر رکورد دارای کلیدی باشد مؤلفه به نام های f_1, \dots, f_k از داده گونه های به ترتیب f_1, \dots, f_k می باشد، باشد. تعداد حالت های f_i را s_i فرض می کیم. در مرتب کردن A پس از اولویت دارده ایم:

$B_i : array[i]$ of list-type برای $(1 \leq i \leq k)$ داریم:

```

Bucket_Sort
1   for i<-k down to 1
2     do for each value v of type t_i
3       do make B_i[v] empty
4     for each record r in list A
5       do move r from A to the end of B_i[v]
6         (* where v is the value of f_i of the key for r *)
7     for each value v of type t_i from lowest to highest
8       do concat B_i[v] to the end of A)

```

اگر هر سطل را بد صورت پک صفت پیاده سازی کیم، هر عمل concat در زمان ثابت قابل اجرا است. زمان اجرا این الگوریتم برابر است با:

$$\sum_{i=1}^k O(s_i + n) = O(kn + \sum_{i=1}^k s_i)$$

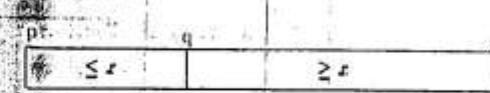
اگر $s_i = n$ آن گاه

$$T(n) = \Theta(kn + \sum_{i=1}^k n) = \Theta(n + kn) = \Theta(n)$$

مثال: فرض کنید کلیدها شامل سه مؤلفه با مقادیر a_1, \dots, a_n و b_1, \dots, b_n هستند و ۷ رکورد زیر داده شده اند.

a_1	a	5	1320
a_2	c	12	1310
a_3	b	12	1305
a_4	a	8	1400
a_5	z	10	1308
a_6	b	12	1304
a_7	a	7	1310

سه سطل (Bucket) برای کلیدها در نظر می گیریم. ابتدا بر اساس $[3]$ داده ها را در سطل مرتب شده ایم. سپس آن ها را به ترتیب از سطل ها برداشته و صفت جدید را تولید می کیم و به همین ترتیب در مورد سطل های دیگر عمل می کیم.



شکل ۳.۲: تقسیم‌بندی در الگوریتم quicksort

	صف خروجی ۱	Bucket[3]	صف خروجی ۲	Bucket[2]	صف خروجی ۳	Bucket[1]	صف خروجی ۴
a_1	a_1, a_2, a_3	a_1	a_1	a_1	a_1, a_2, a_3	a_1	a_1
a_2	a_2	a_2	a_2	a_2	a_2	a_2	a_2
a_3	a_3	a_3	a_3	a_3	a_3	a_3	a_3
a_4	a_1, a_2	a_4	a_4	a_4	a_4	a_4	a_4
a_5	a_1	a_5	a_5	a_5	a_5	a_5	a_5
a_6	a_6	a_6	a_6	a_6, a_7, a_8	a_6	a_6	a_6
a_7	a_7	a_7	a_7	a_7	a_7	a_7	a_7
a_8	a_8	a_8	a_8	a_8	a_8	a_8	a_8

۳.۲ الگوریتم‌های مرتب‌ساز مبتنی بر مقایسه

ا) انتها چیزی که تحقق یک روش را غیر ممکن می‌سازد، ترس از شکست است.

۳.۲.۲ الگوریتم quicksort

quicksort گویا مرتب‌سازی است که بک آرایه n عنصری را در بدترین حالت با $O(n^2)$ و در حالت متوسط (چیزی که معمولاً از آن انتظار داریم) در $O(n \log n)$ مرتب می‌کند. البته ضریب ثابت $n \log n$ کاملاً کوچک است. این الگوریتم حتی برای محیط‌های حافظه مجازی نیز کارآمد است.

توصیف الگوریتم

quicksort مانند merge-sort مبتنی بر روش تقسیم و حل است. فرض کنید می‌خواهیم آرایه $A[p..r]$ را مرتب کنیم. این الگوریتم شامل سه مرحله‌ی زیر است.

۱. تقسیم: آرایه $A[p..r]$ به دو آرایه ناتنهی $A[p..q]$ و $A[q+1..r]$ تقسیم می‌شود. این طوری که هر عنصر زیرآرایه $A[p..q]$ از هر عنصر زیرآرایه $A[q+1..r]$ ستر کاست (یعنی $A[p..q] \leq A[q+1..r]$).

۲. حل: دو زیرآرایه $A[p..q]$ و $A[q+1..r]$ هر کدام به صورت میانی مرتب شوند.

۳. ترکیب: جوی زیرآرایه‌های صورت در مرحله‌ی دیگر شاهد شدند را با آرایه مرتب شده‌است.

بردازی زیر quicksort را می‌دانیم که

partition
in-place

۳.۲ الگوریتم‌های مرتب‌ساز مبتنی بر مقایسه

```
QuickSort(A,p,r)
```

```
1   if p < r
2       then q ← partition(A,p,r)
3           QuickSort(A,p,q)
4           QuickSort(A,q+1,r)
```

برای این که تمام لکت آرایه مرتب شود پردازه به صورت $\text{QuickSort}(A,1,\text{length}(A))$ فراخوانی می‌شود.
پردازه‌ی Partition که تقسیم‌بندی را انجام می‌دهد، پیاده‌سازی‌های مختلف می‌تواند داشته باشد؛ یکی از آن‌ها چنین است:

```
Partition(A,p,r)
```

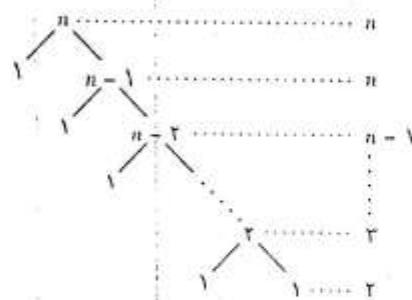
```
1   x ← A[p]
2   i ← p-1
3   j ← r+1
4   while TRUE
5       do repeat j ← j-1
6           until A[j] ≤ x
7           repeat i ← i+1
8           until A[i] ≥ x
9           if i < j
10          then exchange A[i] ↔ A[j]
11          else return j
```

Proof

در این پیاده‌سازی Partition چنین عمل می‌کند. ایندا عنصر اول بدنام x را به عنوان محور^{۱۶} انتخاب می‌کند (در اینجا $A[p]$ اولی محور) می‌تواند هر کدام از عناصر p تا r هم باشد؛ اگر محور $(A[q])$ باشد و برحسب اتفاق بزرگترین عنصر این پردازه در حلقه می‌اندازد گیر خواهد کرد).

بسیار مساوی $i = p - 1$ و $j = r + 1$ قرار می‌دهد (دیده می‌شود که در اولین مرحله i و j اندیس‌های زیرآرایه نبایند ولی این کار گلکنی جالب برای کاهش تعداد عملیات در حلقه‌ی while می‌باشد. حال در حلقه‌ی while دو حلقه‌ی repeat داریم که یکی i را به سمت راست و دیگری j را به سمت چپ حرکت می‌دهد. بدین ترتیب که از روزی عناصر $x < A[i]$ به سمت راست رفته‌اند و اگر $x \geq A[j]$ باشد می‌باشد. هم‌چنین، از روزی عناصر $x > A[i]$ به سمت چپ رفته‌اند و اگر $x \leq A[j]$ باشد مترقب می‌شود. بدینه است که حتماً این دو حلقه‌ی repeat می‌توانند در این صورت، اگر $i < j$ باشد $A[i] < A[j]$ باهم تعویض می‌شوند و حلقه‌ی while تکرار می‌شود تا وقتی که $i \geq j$. توجه کنید که در انتهای، رویه‌ی partition آرایه $A[p..r]$ را به دو قسمت ناهمی تقسیم می‌کند و همچنین از عناصر قسمت اول از همچنین از عناصر قسمت دوم بزرگتر نیست. به این دلیل الگوریتم QuickSort می‌باشد.

pivot^{۱۷}

شکل ۴.۲: درخت بازگشت برای $T(n) = T(n-1) + \Theta(n)$

تحلیل الگوریتم

ابن الگوریتم بدترین حالت از $\Theta(n^2)$ و در حالت متوسط از $\Theta(n \log n)$ می‌باشد و ابن کارایی وابسته به نحوی تقسیم‌بندی است. هرینه‌ی عمل تقسیم‌بندی یک آرایه‌ی به اندازه‌ی n برابر $O(n)$ با ثابت کوچک است. چرا که در الگوریتم ارایه شده، هیچ‌گاه متغیرهای i و j به عقب بر نمی‌گردند و تعداد تغییرها حداقل برابر $n/2$ است.

تقسیم‌بندی در بدترین حالت

بدترین رفتار quicksort هنگامی اتفاق می‌افتد که برداره‌ی Partition در تمام مراحل الگوریتم n عنصر را به $1 - n$ و 1 تقسیم کند در این حالت پیجیدگی زمان الگوریتم به صورت $T(n) = T(n-1) + \Theta(n)$ خواهد

بود که با حل آن بعساندگی دیده می‌شود که $T(n) = \Theta(n^2)$ (شکل ۴.۲).

توجه کنید که در این پیاده‌سازی اگر آرایه از قلی مرتبت باشد باز همیه در بدترین حالت $T(n) = \Theta(n^2)$ می‌باشد (ما \times را برابر $A[p:n]$ قرار می‌دهیم).

تقسیم‌بندی در بهترین حالت

بهترین حالت هنگامی اتفاق می‌افتد که رویه Partition در هر مرحله n عنصر از ابتداء به دو آرایه با تعداد عناصر $\lceil \frac{n}{2} \rceil$ و $\lfloor \frac{n}{2} \rfloor$ تقسیم کند. پیجیدگی الگوریتم در این حالت چنین محاسبه می‌شود:

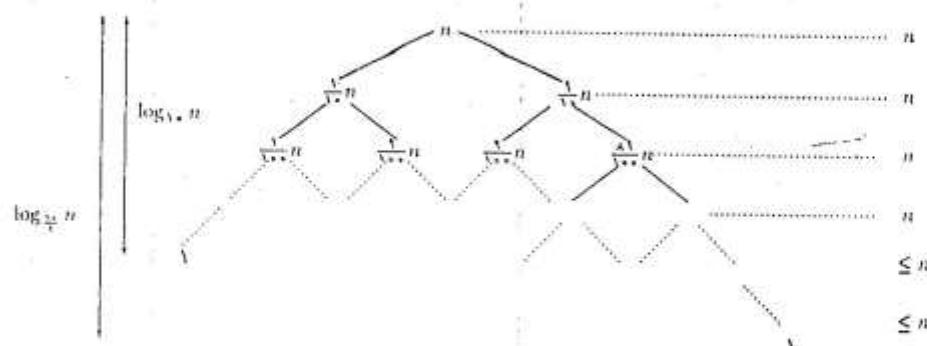
$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n) \rightarrow T(n) = \Theta(n \log n)$$

تقسیم‌بندی متوازن

همان طور که بعد از شان داده خواهد شد حالت متوسط به بهترین حالت نزدیکتر است نایابترین حالت. کلید فهم این مطلب در این است که متوازن چگونه در رابطه بازگشتی معکوس می‌شود. برای مثال این الگوریتم تقسیم کننده را در نظر بگیرید که همیشه آرایه را به نسبت 1 به 1 تقسیم می‌کند. در نگاه اول این الگوریتم کاملاً نامتوازن به نظر می‌رسد. ولی چنین نیست! رابطه‌ی بازگشتی آن $T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + n$ می‌باشد که حل آن

$T(n) = O(n \log n)$. درخت بازگشت این رابطه بازگشتی در شکل ۵.۲ نشان داده می‌شود. توجه کنید که حتی اگر این نیست 1 باشد (و در حالت کلی نا وتنی که همواره کسری، و نه به اندازه‌ی ثابت، از تعداد عناصر در بخش‌ها تفاضل بزرگ است) نیز الگوریتم از مرتبه‌ی $n \log n$ خواهد بود.

نهادی برای حالت متوسط

شکل ۵.۲: درخت بارگشت برای $T(n)$

انتظار ما از تابع **Partition** این است که هم تقسیم‌های خوب و هم بد انجام بدهد و این تقسیم‌ها به طور تصادفی در درخت بارگشت پراکنده شوند و معلوم است (!) که این تقسیم خوب و بد از تقسیم ۹ به ۱ مثال بالا بهتر است، پس این انتظار ما هم که حالت متوسط به بهترین حالت نزدیک‌تر باشد تا به بدترین حالت، جندان غیر معقول است! برای تحلیل الگوریتم در حالت متوسط، گونه‌ی تصادفی آن را در نظر می‌گیریم.

۲.۳.۲ گونه‌ی تصادفی quicksort

در بررسی رفتار quicksort در حالت متوسط ما فرض کردیم که همه‌ی جای‌گشت‌های عناصر ورودی با شناسی پکان ظاهر می‌شوند. اگر این فرض در مورد داده‌ی ورودی درست باشد، از نظر متوسط زمان اجرا، quicksort یک الگوریتم بهینه‌ی مرتب‌ساز است **randomized**.
یک الگوریتم را «صادفی»^{۱۷} گوییم که علاوه بر آن که رفتار آن توسط ورودی مشخص می‌شود، رفتار آن به مقداری که مولده عدد تصادفی تولید می‌کند هم وابسته باشد. در تحلیل الگوریتم فرض می‌کنیم یک تابع Random(a,b) داریم که به صورت تصادفی آماری یک عدد بین a و b تولید می‌کند.
در گونه‌ی تصادفی quicksort قبل از جدا زدن تابع **Partition** : $A[p : r]$ را با یک عدد تصادفی بین p و r عوض می‌کنیم.

```

Randomized_Partition(A,p,r)
1   i <- Random(p,r)
2   exchange A[p] <-> A[i]
3   return Partition(A,p,r)

```

randomized^{۱۷}

```

Randomized_QuickSort(A,p,r)
1   if p < r
2       then q<-Randomized_Partition(A,p,r)
3           Randomized_QuickSort(A,p,q)
4           Randomized_QuickSort(A,q+1,r)

```

در بخش بعدی که کارایی الگوریتم را بررسی می کنیم از این الگوریتم استفاده خواهیم کرد.

تحلیل پیچیدگی زمان quicksort

در بحث فعلی گفتیم که بدترین حالت quicksort هنگامی رخ می دهد که تابع Partition آرایه n عنصری را به یک زیرآرایه 1 عضوی و بک زیرآرایه $n - 1$ عضوی تقسیم کند که در این حالت پیچیدگی الگوریتم $\Theta(n^2)$ است. این ادعا را ثابت می کنیم. برای این کار نشان می دهیم که $T(n) = \Omega(n^2)$ و نیز $T(n) = O(n^2)$. پارامتر $T(n) = \max\{T(q) + T(n-q)\} + \Theta(n)$. فرض کنید $T(n) = \Omega(n^2)$ رساند. $T(n) = \max\{T(q) + T(n-q)\} + \Theta(n)$ آرایه n عضوی را به یک زیرآرایه $n - q$ عضوی تقسیم کند. در این صورت، $T(n) = \max\{T(q) + T(n-q)\} + \Theta(n)$. چون دو ناحیه ای ایجاد شده هر کدام حداقل یک عضو دارند. با استفاده از این می کنیم که $T(n) = \Omega(n^2)$. اگر برای $m < n$ فرض کنیم $T(m) \leq cn^2$ داریم

$$T(n) \leq \max\{cq^2 + c(n-q)^2\} + \Theta(n)$$

بیشینه $cq^2 + (n-q)^2$ را در نظر گرفتن $1 \leq q \leq n-1$ داریم:

$$T(n) \leq c(1 + (n-1)^2) + \Theta(n) = cn^2 - 2c(n-1) + \Theta(n)$$

$$T(n) \leq cn^2$$

به شرطی که c را آنقدر بزرگ بگیریم که $\Theta(n)$ را بیوشاند. حال نشان می دهیم که $T(n) = \Omega(n^2)$.

$$T(n) \geq cn^2, m \leq n \rightarrow T(n) \geq \max_{1 \leq q \leq n-1} \{cq^2 + c(n-q)^2\} + \Theta(n)$$

$$T(n) \geq cn^2 - m^2 \rightarrow T(n) \geq \max_{1 \leq q \leq n-1} \{(n-q)^2\} + \Theta(n)$$

پس در حالت ۱ داریم:

$$T(n) \geq cn^2 - 2c(n-1) + \Theta(n) \geq cn^2$$

به شرطی که c آنقدر کوچک باشد که $\Theta(n) - 2c(n-1)$ مثبت شود. پس

$$T(n) = \Omega(n^2) = O(n^2) \Rightarrow T(n) = \Theta(n^2)$$

و ادعا ثابت می شود.

برای حالت متوسط، نوجه کنید که همراه محور عنصر اول آرایه A است که به طرز تصادفی انتخاب می شود. مرتبهی محور نسبت به n عنصر A می تواند هیچ از ۱ تا n باشد. اگر احتمال این حالات را بگسان

۳.۲ الگوریتم‌های مرتب‌ساز مبتنی بر مقایسه

پنجم، احتمال این که مرتبه‌ی محور k باشد برابر $\frac{1}{n}$ است. حال توجه کنید که برای هر مرتبه‌ی $n \leq k \leq n - 1$ آرایه A به چه نحو تقسیم می‌شود. در هر دو حالت $1 \leq k = 2$ و $1 \leq k \leq n - 1$ آرایه‌ی A به دو قسمت با اندازه‌های 1 و $n - 1$ تقسیم می‌شود. (چرا؟) بنابراین اگر $\hat{T}(n)$ متوسط زمان اجرای الگوریتم Randomized-QuickSort باشد، داریم

$$\hat{T}(n) = \frac{1}{n} \left[\hat{T}(1) + \hat{T}(n-1) + \sum_{q=1}^{n-1} [\hat{T}(q) + \hat{T}(n-q)] \right] + \Theta(n) \quad (1)$$

در بحث‌های قبلی دیدیم که در بدترین حالت $\hat{T}(n) \leq T(n)$ است و $T(1) = \Theta(1)$ و نیز $T(n-1) = \Theta(n^\tau)$ داریم.

$$\frac{1}{n} [\hat{T}(1) + \hat{T}(n-1)] \leq \frac{1}{n} [\Theta(1) + \Theta(n^\tau)] = \Theta(n) \quad \text{بس!}$$

بنویه به فرمول ۳ داریم:

$$\hat{T}(n) = \frac{1}{n} \sum_{q=1}^{n-1} [\hat{T}(q) + \hat{T}(n-q)] + \Theta(n) = \frac{\gamma}{n} \sum_{k=1}^{n-1} \hat{T}(k) + \Theta(n)$$

برای حل این رابطه‌ی بازگشتی، حدس می‌زنیم $\hat{T}(n) \leq an \log n + b$ و نلاش می‌کنیم حدس مان را ثابت کنیم.

$$\begin{aligned} \hat{T}(n) &\leq an \log n + b, \quad a, b > 0 \\ \hat{T}(n) &= \frac{\gamma}{n} \sum_{k=1}^{n-1} \hat{T}(k) + \Theta(n) \\ &= \frac{\gamma}{n} \sum_{k=1}^{n-1} ak \log k + b + \Theta(n) \\ &= \frac{\gamma}{n} \sum_{k=1}^{n-1} ak \log k + \frac{\gamma}{n} \sum_{k=1}^{n-1} b + \Theta(n) \\ &= \frac{\gamma a}{n} \sum_{k=1}^{n-1} k \log k + \frac{\gamma b}{n} (n-1) + \Theta(n) \end{aligned}$$

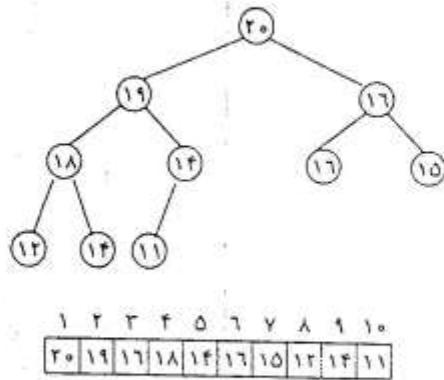
و از نزدیک انتگرال ثابت می‌شود که:

$$\sum_{k=1}^{n-1} k \log k \leq \frac{1}{\gamma} n^\tau \log n - \frac{1}{\lambda} n^\tau$$

از این نامساوی و رابطه‌ی بالا داریم:

$$\begin{aligned} \hat{T}(n) &\leq \frac{\gamma a}{n} \left(\frac{n^\tau}{\gamma} \log n - \frac{n^\tau}{\lambda} \right) + \frac{\gamma b}{n} (n-1) + \Theta(n) \\ &\leq an \log n + b + [\Theta(n) + b - \frac{an}{\gamma}] \end{aligned}$$

$\hat{T}(n) \leq an \log n + b$ را طوری انتخاب می‌کنیم تا درون پراکنده شود. بنابراین



شکل ۶.۲: تبادل درختی و آرایه‌ی یک heap

۴.۲ الگوریتم HeapSort

«ناریکرین ساعت شب، یک ساعت قبل از طلوع آفتاب است.^{۱۸}»
 یک درخت دودویی کامل (به جز خداکتر یک گره) و متوالن (یعنی این که سطح برگ‌های آن حداقل یک واحد اختلاف دارد) است که برگ‌های سطح آخر آن از سمت چپ چند شده‌اند. هم‌چنین کلید هر عنصر از کلیدهای فرزندانش کوچکتر نمی‌باشد، یعنی $A[i] \geq A[\text{parent}(i)]$ است. از سمت چپ چند شده‌اند. هم‌چنین کلید هر عنصر از در این تبادل فرزند چپ عنصر نام در $A[2i]$ (اگر $n \leq 2i$) و فرزند راست آن در $A[2i+1]$ (اگر $n \leq 2i+1$) و پدرش در $A[\frac{i}{2}]$ (اگر $i \neq 1$) قرار دارد.

۱.۴.۲ ویژگی‌های یک heap

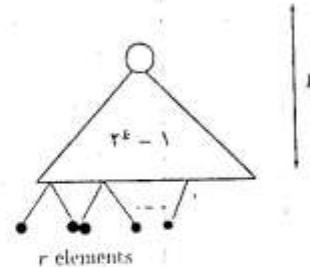
لم ۱. ریشه بزرگترین عنصر heap است.

اینها بر روی ارتفاع درخت به سادگی می‌توان ثابت کرد که ریشه‌ی هر یک درخت بزرگترین عنصر آن زیردرخت است. سیارا یک گره‌ی تنها در نظر می‌گیریم. \square

لم ۲. ارتفاع یک heap با n عنصر $= \lceil \log n \rceil$ می‌باشد.

اینها. طبق تعریف، درخت ناسطح $1 - h$ است. همچنین برای هر یک n موجود است به طوری که $n < 2^{4+r}$ و $n = 2^k + r$ که در آن $2^k \leq n \leq 2^{k+1}$. این r که با این تعداد عنصر می‌سازیم شامل

^{۱۸} یک مرتب‌نمای اسپایرونی.

شکل ۴.۲: ارتفاع یک heap با n عنصر.

یک درخت دودویی به برد ارتفاع $1 - k$ و یا $1 - 2^k$ عنصر است که به آن یک درست k از سمت چپ اضافه شده است (شکل ۴.۲)، واضح است که ارتفاع heap برابر $\lceil \log n \rceil$ می‌باشد.

اگر در تعریف بالا شرط ناکوچکتر را به تابع Heapify تبدیل کیم، heap به صف اولویت (priority queue) تبدیل می‌شود که در انتهای این بخش به آن اشاره خواهد شد.

در الگوریتم HeapSort از چند روشی مهم به شرح زیر استفاده شده است:

:Heapify

وروی این پردازه آرایه A و اندیس مربوط به یک عنصر از آرایه است. با فرض آن که خاصیت heap برای زیردرختان چپ و راست عنصر ابرقرار است، زیردرخت به ریشه‌ی ارا به صورت heap در می‌آورد.

```

    Heapify(A,i)
1      l <- left(i)
2      r <- right(i)
3      if(l <= n) and (A[l] > A[i])
4          then largest <- l
5      else largest <- i
6      if(r <= n) and (A[r] > A[largest])
7          then largest <- r
8      if(largest <- i)
9          then swap(A[i], A[largest])
10         Heapify(A, largest)

```

اندازه‌ی زیردرخت‌های یک heap با n عنصر حداقل $\frac{n}{2}$ است. این زمانی اتفاق می‌افتد که زیردرخت سمت چپ یک درخت به ارتفاع h و زیردرخت سمت راست یک درخت به ارتفاع $1 - h$ باشد. در نتیجه $T(n)$ زمان اجرای Heapify برابر است با:

$$T(n) \leq T\left(\frac{n}{2}\right) + \theta(1)$$

فصل ۲ مرتب‌سازی و مرتبه‌ی آماری

۴۲

و طبق قضیه‌ی اصلی داریم: $T(n) = O(\log n)$. البته بدترین حالت هنگامی روی می‌دهد که روید نا رسیدن به یک بزرگ ادامه پیدا کند. بنابراین $T(n) = \Omega(\log n)$ و از این دو تبیجه می‌گیریم که $T(n) = \Theta(\log n)$. اگر $S(n)$ حداکثر تعداد تعریض‌ها در الگوریتم فوق باشد، داریم:

$$\begin{aligned} S(n) &= \sum_{i=1}^n [1 + \lfloor \log i - 1 \rfloor] \\ &< n + \sum_{i=1}^n \log i = n + n \log n \end{aligned}$$

:Build-Heap

این پردازه یک آرایه A با $n = \text{length}(A)$ را با استفاده از Heapify به یک Heap تبدیل می‌کند. با توجه به این که درایه‌های $A[i]$ برای $i=1\dots n/2$ هر کدام هرکدام یا یک عنصر (همان $A[i]$) است، کار $\text{Heapify}(A[i], \lfloor n/2 \rfloor)$ ۱ دنبال می‌شود.

```
Build_Heap(A)
1   heap_size(A) <- length(A)
2   for i <- [length(A)/2] down to 1
3       Heapify(A,i)
```

حداکثر زمان اجرای این پردازه برابر است با:

$$T(n) = O(n)O(\log n) = O(n \log n)$$

اما زمان اجرای واتی این کمتر است.

لم ۳. حداکثر تعداد گره‌های با ارتفاع h در یک heap یا عنصر برابر است با $\lceil n/2^h \rceil$.

این باتوجه به اینکه عدد عناصر با اندیس i فرزنده ندارند، یعنی $2i > n$ ، فرض کنید n_h تعداد گره‌های در ارتفاع h در درخت T با n گره باشد و نیز آن که اعداً برای گره‌های با ارتفاع h درست باشد. اگر T' با n' همان T باشد که برگ‌های آن را برداشته باشیم، داریم $n' = n - \lceil n/2 \rceil = \lceil n/2 \rceil$. هم‌چنین می‌دانیم که گره‌هایی که ارتفاعشان در T برابر h است، در T' در ارتفاع $1-h$ قرار دارند. بنابراین:

$$n_h = n'_{h-1} \leq \lceil n'/2^h \rceil = \lceil (n/2)/2^h \rceil \leq \lceil (n/2)/2^h \rceil = \lceil n/2^{h+1} \rceil.$$

و حکم ثابت است.

می‌دانیم که هزینه‌ی Heapify بر روی گره‌های با ارتفاع h از $O(h)$ می‌باشد. با توجه به این که $h \leq \log n$ ، داریم:

$$T(n) = \sum_{h=1}^{\lfloor \log n \rfloor} \lceil \frac{n}{2^{h+1}} \rceil O(h)$$

۴۲



شکل ۴.۲: ساختار تعداد تعویض‌ها.

$$= O\left(\sum_{h=1}^{\lfloor \log n \rfloor} h \frac{1}{\tau^h}\right)$$

$$\sum_{h \geq 1} \frac{h}{\tau^h} = \frac{\frac{1}{\tau}}{\left(1 - \frac{1}{\tau}\right)^2} = \tau$$

از آنجایی که
(۱)
بنابراین:

$$T(n) = O\left(\sum_{h \geq 1} \frac{h}{\tau^h}\right) = O(n)$$

این اثبات دیگر برای این نتیجه به صورت زیر است^{۱۹}. اگر حداقل تعداد تعویض‌های $S(i)$ را `Heapsify`(A, i) می‌نواند که به این معنی با توجه به شکل ۴.۲ می‌توان دید که

$$S(i) = \begin{cases} 1 & i = \lfloor n/2^k \rfloor + 1 \quad \text{تعداد} = \lfloor n/2^k \rfloor \\ 2 & i = \lfloor n/2^k \rfloor + 1, \dots, \lfloor n/2^k \rfloor \quad \text{تعداد} = \lfloor n/2^k \rfloor \\ \dots & \dots \\ k & i = \lfloor n/2^{k-1} \rfloor + 1, \dots, \lfloor n/2^k \rfloor \quad \text{تعداد} = \lfloor n/2^{k-1} \rfloor \\ \dots & \dots \\ \lfloor \log n \rfloor & i = 1 \quad \text{تعداد} = 1 \end{cases}$$

زمان اجرا مناسب است با حداقل تعداد کل تعویض‌ها:

$$T(n) \leq \sum_{k=1}^{\lfloor \log n \rfloor} k \frac{n}{\tau^{k+1}}$$

که با استفاده از فرمول ۲ داریم $T(n) = O(n)$.

:Heap-sort

در این روش ابتدا `heap` ساخته می‌شود. سپس روشهای در دراین اول قرار دارد ($A[1]$) و بزرگترین عنصر آرایه است با آخرین عنصر $A[n]$ تعویض می‌شود. با این کار مبارگه بین عناصر در جای نهایی خود قرار می‌گیرد و از اندازه `heap` پکی کم می‌شود. در این مرحله پردازشی `Heapsify` بر روی عنصر اول آرایه و اندازه ۱ - n برقراری دوباره‌ی خاصیت `heap` اجرا می‌شود. الگوریتم به دوین مرتبه ارتکار می‌شود تا همه‌ی عناصر مرتب شوند.

^{۱۹} این اثبات در کلاس آرایه شد.

```

    HeapSort(A)
1      Build_Heap(A)
2      for i <= length(A) downto 2
3          do exchange A[1] <-> A[i]
4              heap_size(A) <-> heap_size(A)-1
5              Heapify(A,1)

```

برای محاسبه زمان اجرای الگوریتم، توجه داریم که ساخته‌ای اصلی الگوریتم ۱ - «بارنکار می‌شود و هر بار یک جابجاپی و یک بار فراخوانی Heapify انجام می‌شود، در نتیجه:

$$T(n) = O(n) + (n-1)O(\log n) = O(n \log n)$$

می‌توان نشان داد که زمان اجرا $\Omega(n \log n)$ می‌باشد (تحلیل کرد).

۵.۲ میانه‌ها و مرتبه‌های آماری

مرتبه‌ی آماری i ام از یک مجموعه n عضوی، i -امین عضو کوچکتر این مجموعه می‌باشد. برای مثال، کمینه i ای یک مجموعه مرتبه‌ی آماری یک این مجموعه است، و بیشینه i ای یک مجموعه مرتبه‌ی آماری i ام این مجموعه می‌باشد ($i = n$).

یک میانه به صورت شهودی «عضو وسطی» مجموعه می‌باشد. وقتی «فرد باشد، میانه یکنائب و مرتبه‌ی آماری $\frac{n}{2} + 1$ است، وقتی n زوج باشد، دو میانه داریم که مرتبه‌های آماری i و $i + 1$ باشند، وقتی $n/2 + 1$ باشد، میانه‌ها مرتبه‌های آماری $\lceil \frac{n+1}{2} \rceil$ و $\lfloor \frac{n+1}{2} \rfloor$ باشند. بنابراین بدون در نظر گرفتن زوجیت n ، میانه‌ها مرتبه‌های آماری $\lceil \frac{n+1}{2} \rceil$ و $\lfloor \frac{n+1}{2} \rfloor$ باشند.

در این بخش انتخاب مرتبه‌ی آماری i ام از یک مجموعه با n عدد متفاوت را مورد بررسی قرار می‌دهیم. با این که برای راحتی اعداد را متفاوت در نظر گرفته، تایمی نه در اینجا، بدست می‌آوریم به صورت شهودی قابل تعمیم به حالتی که اعداد تکراری داریم نیز هستند، مسئله انتخاب i را می‌توان به صورت زیر بیان کرد:
problem
روزدی: یک مجموعه A از n عدد (متفاوت) و یک عدد i که داریم $i \leq n$ باشد.
خروجی: عضو $A \in i$ که دفیغاً از i - ا عضو در A بزرگ‌تر است.

مسئله انتخاب را می‌توان در $O(n \lg n)$ حل کرد. به این ترتیب که ابتدا اعداد را با استفاده از الگوریتم heapsort یا mergesort مرتب می‌کنیم و در این صورت عنصر i ام در آرایه مرتب شده جواب مسئله است. ولی الگوریتم‌های سرعی تری نیز وجود دارند.

مالیتا مسئله‌ی یافتن هر زمان کوچک‌ترین و بزرگ‌ترین عنصر یک مجموعه را بررسی می‌کنیم. بعد از حل مسئله انتخاب می‌پردازیم. در قسمت دوم یک الگوریتم با زمان متوسط $O(n)$ را بررسی می‌کنیم و بخش سوم شامل یک الگوریتم با دیدگاهی نظری است که این مسئله را در یادربی حالت در $O(n)$ حل می‌کند.

```

under statistics
minimum
maximum
selection problem

```

۱.۰.۲ کمینه و بیشینه

چند مقایسه برای یافتن کمینه یک مجموعه «عضوی لازم است؟ ما به راحتی می‌توانیم حد بالای ۱ - «را برای تعداد مقایسه‌ها پیدا کنیم؛ اعداد مجموعه را یک به یک اسنجان می‌کنیم و کوچکترین عضوی را که تاکنون دیده‌ایم را نگاه می‌داریم. در این روش ما فرض کردی‌ایم که مجموعه درون آرایه A ، ذخیره شده است، که در آن $\text{length}[A] = n$.

```

Minimum(A)
1 min ← A[1]
2 for i ← 2 to length[A]
3   do if min > A[i]
4     then min ← A[i]
5 return min

```

الته، پیدا کردن بیشینه بیز با n مقایسه میسر است. آیا این بهترین راه حل است؟ بله، برای این که ما می‌توانیم یک حد پایین از ۱ - « مقایسه را برای یافتن کمینه بدست آوریم. الگوریتمی را تصور کنید که یافتن کمینه را به صورت یک دوره مسابقه بین عناظر در نظر می‌گیرد. هر مقایسه یک مسابقه است که در آن عضو کوچکتر پیروز می‌شود. تکنیک لکلیدی این است که همهی عناظر به جز برندگی نهایی باید حافظل در یک مسابقه شکست خورده باشند. بنابراین، حافظل ۱ - « مقایسه برای یافتن کمینه لازم است، و الگوریتم Minimum از نظر تعداد کل مقایسه‌ها بهترین است.

یک تکنیک جالب در تحلیل این الگوریتم، یافتن متوسط تعداد اجرا شدن خط شماره‌ی چهار است. [مسئله‌ی در کتاب از شما می‌خواهد ناشان دهید این مقدار $\Theta(\lg n)$ است.]

یافتن همزمان بیشینه و کمینه

در برخی از کاربردها، ما باید بیشینه و کمینه یک مجموعه n عضوی را باهم پیدا کنیم. برای مثال، یک برنامه‌ی گرافیکی ممکن است بیاز داشته باشد تا یک سری داده‌های به صورت $((x, y))$ را طوری مفاسی کند که درون یک صفحه‌ی نمایش یا هر دستگاه گرافیکی مستطیل شکل دیگر قرار بگیرند. برای انجام این کار، برنامه‌ای باید بیشینه و کمینه را در هر مختصه پیدا کند.

یافتن الگوریتمی که بتواند بیشینه و کمینه از n عنصر را به طور همزمان در تعداد مقایسه‌هایی از مرتبه‌ی (n) که از نظر آماری بهینه بیز هست پیدا کند، چنان مشکل نیست. کافی است تا بیشینه و کمینه را به صورت جدا از هم، و هر کدام را با $1 - n$ مقایسه پیدا کنیم؛ و در مجموع $2 - n$ مقایسه انجام داده‌یم.

در حقیقت، تنها $2\lceil n/2 \rceil$ مقایسه برای یافتن همزمان کمینه و بیشینه لازم است. برای انجام این کار، ما عناظر بیشینه و کمینه‌ای که تا به حال دیده‌ایم را مدیریت می‌کنیم. بحای برداش هر عضو مجموعه ورودی با روند یک مقایسه با بیشینه‌ی فعلی و یک مقایسه با کمینه‌ی فعلی، که در مجموع هزینه‌ای برای دو مقایسه برای هر عنصر را دارد، ما عناظر را جفت جفت برداش می‌کنیم. ما اینتا دو عنصر ورودی را با یکدیگر مقایسه می‌کنیم، و سپس عنصر کوچکتر را با کمینه‌ی فعلی و عنصر بزرگتر را با بیشینه‌ی فعلی مقایسه می‌کنیم، که با هزینه‌ی سه مقایسه به ازای هر دو عنصر انجام می‌شود.

تمرین ها

تمرین ۱-۱

نشان دهد که دو مین کمینه از میان «عنصر را می‌توان در بدترین حالت با $-2 - \lceil \lg n \rceil + n$ مقایسه یافت.

(راهنمایی: کمینه را هم پیدا کنید.)

تمرین ۲-۱ *

نشان دهد که در بدترین حالت، $2n/2$ مقایسه برای یافتن هم‌زمان بیشینه و کمینه از میان n عدد ضروری است.

(راهنمایی: در نظر بگیرید که کدام اعداد دارای استعداد کمینه بودن و کدام دارای استعداد بیشینه بودن هستند و

نشان دهد که هر مقایسه چگونه بر روی این‌ها تاثیر می‌گذارد.)

۲.۵.۲ انتخاب در زمان متوسط خطی

در وله‌ی اول حل مسئله‌ی عمومی انتخاب^{۲۲} بسیار مشکل‌تر از مسئله‌ی یافتن کوچکترین عنصر به نظر می‌رسد، ولی، به طور غیر منظره‌ای، مزینه‌ی زمان اجرای هردو مسئله باهم برابر است: $\Theta(n)$. در این بخش ما پک الگوریتم تقسیم و حل برای مسئله‌ی انتخاب آرایه می‌دهیم. الگوریتم Randomized-Select مشابه الگوریتم quicksort در فصل ۲ می‌باشد. آبده‌ی ما، همانند quicksort، تقسیم آرایه ورودی به صورت بازگشتی است. ولی برخلاف quicksort، که به صورت بازگشتی بر روی هر دو طرف نقطه‌ی تقسیم فراخوانده می‌شود، Randomized-Select تنها بر یک طرف نقطه‌ی تقسیم عمل می‌کند. این تقسیت در بررسی زمان اجرا خود را نشان می‌دهد: با وجود این که زمان اجرای متوسط $\Theta(n \lg n)$ از quicksort است، زمان اجرای متوسط $\Theta(n)$, Randomized-Select است.

از رویه‌ی Randomized-Partition که در فصل ۸.۳ کتاب معرفی شده است استفاده می‌کند. بنابراین، مشابه Randomized-Quicksort، یک الگوریتم تصادفی است، زیرا بخشی از رفتار آن با استفاده از خروجی تولید‌کننده‌ی عدد تصادفی تعیین می‌شود، برنامه‌ی زیر برای Randomized-Select نامین کمینه را در آرایه $A[p..r]$ بر می‌گرداند.

<pre> Randomized-Select(A, p, r, i) 1 if $p = r$ 2 then return $A[p]$ 3 $q \leftarrow$ Randomized-Partition(A, p, r) 4 $k \leftarrow q - p + 1$ 5 if $i \leq k$ 6 then return Randomized-Select(A, p, q, i) 7 else return Randomized-Select($A, q + 1, r, i - k$) </pre>	$\text{Randomized-Partition}(A, p, r)$ $\begin{cases} \text{if } (p=r) \\ \text{then return } A[p] \\ q \leftarrow \text{Randomized-Partition}(A, p, r) \\ k \leftarrow r-p+1 \\ \text{if } i \leq k \\ \text{then return } A[q] \\ \text{else return } A[q+1..r] \end{cases}$
---	--

بعد از این که Randomized-Partition در خط سوم الگوریتم اجرا می‌شود، آرایه $A[p..r]$ دو زیرآرایه غیر تهم $[q..r]$ و $[p..q-1]$ دارد. تقسیم می‌شود به طوری که هر عضو $A[p..q-1]$ از هر عضو $A[q..r]$ کوچکتر است. خط چهارم در الگوریتم عدد k یعنی تعداد عناصر در زیرآرایه $A[q..r]$ را محاسبه می‌کند. حال الگوریتم تشخیص می‌دهد که نامین کمینه در کدام‌یک از زیرآرایه‌های $A[p..q-1]$ و $A[q..r]$ است. اگر $k \leq i$ ، آن‌گاه عنصر دلخواه در قسمت پایینی نقطه‌ی تقسیم است. و در خط ششم به صورت بازگشتی از زیرآرایه مذکور انتخاب می‌شود. اما اگر $i > k$ باشد، آن‌گاه عنصر دلخواه در بخش بالایی نقطه‌ی تقسیم است. و چون ماتنکنون k عنصر از $A[p..r]$ که کوچکتر

selection^{۲۳}

از نامن کمینه هستند را می‌شناسیم – در واقع عناصر $[q, i, k]$ ، پس عنصر دلخواه ما $(k - i)$ نامن عنصر کمینه در $[q + 1, n]$ است، که به صورت بازگشته در خط هفتم پیدا می‌شود.

بدترین زمان اجرای الگوریتم Randomized-Select از مرتبه $\Theta(n^2)$ است، و حتا برای یافتن کمینه نیز این مطلب درست است، زیرا که ممکن است ماسیار بدشناس بانش و همیشه نقطه‌ی تقسیم در کنار بزرگترین عنصر آرایه‌ی فعلی باشد، گرچه به خاطر تصادفی بودن این الگوریتم، هیچ ورودی خاصی به این بدی نخواهد برد.

ما می‌توانیم یک حد بالا برای $T(n)$ ، زمان اجرای متوسط Randomized-Select، روی آرایه‌ای از n عنصر را به صورت زیر پیدا کنیم. ما در فصل ۸.۴ دیدیم که الگوریتم Randomized-Partition یک تقسیم تولید می‌کند که بخش کوچکتر آن به احتمال $2/n$ دارای ۱ عنصر است و به احتمال $1/n$ دارای i عنصر است که $i = 2, 3, \dots, n-1$. با فرض اینکه یک دنباله‌ی یکنواخت صعودی باشد، بدترین حالت Randomized-Select همیشه بدشناس است، یعنی i نامن کمینه در بخش بزرگتر نقطه‌ی تقسیم می‌افتد. بنابراین، ما به رابطه‌ی بازگشته زیر می‌رسیم:

$$\begin{aligned} T(n) &\leq \frac{1}{n} \left(T(\max(1, n-1)) + \sum_{k=1}^{n-1} T(\max(k, n-k)) \right) + O(n) \\ &\leq \frac{1}{n} \left(T(n-1) + \frac{1}{\tau} \sum_{k=\lceil n/\tau \rceil}^{n-1} T(k) \right) + O(n) \\ &= \frac{1}{\tau} \sum_{k=\lceil n/\tau \rceil}^{n-1} T(k) + O(n). \end{aligned}$$

خط دوم از خط اول به واسطه‌ی $\max(1, n-1) = n-1$ تتجه می‌شود.

$$\max(k, n-k) = \begin{cases} k & \text{if } k \geq \lceil n/\tau \rceil, \\ n-k & \text{if } k < \lceil n/\tau \rceil. \end{cases}$$

اگر n فرد باشد، آنگاه هر کدام از جمله‌های $(1 - 1), T(\lceil n/\tau \rceil), T(\lceil n/\tau \rceil + 1), \dots, T(n-1)$ دوبار در مجموع ظاهر می‌شوند، و اگر n زوج باشد هر کدام از جمله‌های $(1 - 1), T(\lceil n/\tau \rceil + 1), T(\lceil n/\tau \rceil + 2), \dots, T(n-1)$ دوبار و جمله‌ی $T(\lceil n/\tau \rceil)$ یک بار ظاهر می‌شوند. در هر دو حالت مجموع خط اول از بالا به مجموع خط دوم محدود می‌باشد. خط سوم از خط دوم نتیجه می‌شود البته با استفاده از $O(n^2) = O(n^2)$ ، و بنابراین جمله‌ی $(1 - 1)$ را می‌توان در مقابل $O(n)$ نادیده گرفت.

ما رابطه‌ی بازگشته را با جای‌گذاری حل می‌کنیم، فرض کنید که برای ثابتی مانند c ، داشته باشیم $T(n) \leq cn$ که شرایط اولیه‌ی رابطه‌ی بازگشته را برآورده سازد. با استفاده از فرض استقرا خواهیم داشت

$$\begin{aligned}
 T(n) &\leq \frac{\tau}{n} \sum_{k=\lceil n/\tau \rceil}^{n-1} ck + O(n) \\
 &\leq \frac{\tau c}{n} \left(\sum_{k=1}^{n-1} k - \sum_{k=1}^{\lceil n/\tau \rceil - 1} k \right) + O(n) \\
 &= \frac{\tau c}{n} \left(\frac{1}{\tau}(n-1)n - \frac{1}{\tau} \left(\left[\frac{n}{\tau} \right] - 1 \right) \left[\frac{n}{\tau} \right] \right) + O(n) \\
 &\leq c(n-1) - \frac{c}{n} \left(\frac{n}{\tau} - 1 \right) \left(\frac{n}{\tau} \right) + O(n) \\
 &= c \left(\frac{\tau}{\tau} n - \frac{1}{\tau} \right) + O(n) \\
 &\leq cn
 \end{aligned}$$

زیرا ما می‌توانیم c به قدر کافی بزرگ انتخاب کنیم به طوری که $(1/2)(c/n)^2 + 1/2$ از $O(n)$ پیشی بگیرد.
بنابراین هر مرتبه‌ی آماری، در حالت خاص میانه، را می‌توان با زمان متوسط خطی پاق.

تمرین‌ها

۱-۲

یک نسخه‌ی غیربازگشتی از Randomized-Select را بنویسید.

۲-۲

فرض کنید که ما از Randomized-Select برای یافتن عنصر کمینه در آرایه‌ی $A = \{3, 2, 9, 0, 7, 5, 4, 8, 6, 1\}$ استفاده کنیم. دنباله‌ای از تقسیم‌های متوالی را مشخص کنید که به بدترین حالت کارآمد Randomized-Select منتهی می‌شود.

۳-۲

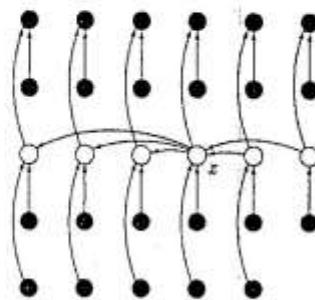
بیان بناورید که در صورت وجود عناصر برابر، رویه‌ی Randomized-Partition زیرآرایه‌ی $A[p, r]$ را به دو زیرآرایه‌ی ناتنهی $A[p, q]$ و $A[q+1, r]$ تقسیم می‌کند به طوری که هر عنصر از $A[p, q]$ کوچک‌تر یا مساوی هر عنصر از $A[q+1, r]$ است. اگر عناصر برابر موجود باشند، آیا Randomized-Select درست کار می‌کند؟

۲.۵.۲ انتخاب در بدترین حالت زمانی خطی

حال مایک الگوریتم انتخاب را بررسی می‌کنیم که زمان اجرای آن در بدترین حالت $O(n^2)$ است. مانند Randomized-Select، الگوریتم Select عنصر دلخواه را با تقسیم بارگشته‌ی آرایه ورودی پیدا می‌کند. ولی ایندهی اساس این الگوریتم، تضمین یک تقسیم خوب در هنگام تقسیم شدن آرایه است. Select از الگوریتم تقسیم نقطی Partition (فصل ۸.۱ رایج‌بیند) استفاده می‌کند، البته با کمی تغییر به طوری که عنصری را که حول آن تقسیم انجام خواهد شد را به عنوان پارامتر ورودی بگیرد:

دهدی می‌بخشته بزرگ حارمه‌ی ارجاع با ارجاع n را در درون دیار آرا مینمی‌کر رصنعن بیما ام تبروگلا

۱. عنصر آرایه‌ی ورودی را به $\lfloor n/5 \rfloor$ گروه هر کدام شامل ۵ عنصر و حداقل یک گروه از $n \bmod 5$ عنصر باقیمانده تقسیم کنید.

شکل ۰.۲: انتخاب با زمان $O(n)$.

۲. عنصر میانه‌ی هر $[n/5]$ گروه، را با اجرای الگوریتم insertion sort روی عناصر هر گروه پیدا کرد. (اگر گروه دارای تعداد زوجی عنصر بود، بزرگترین میانه را انتخاب کرد).

۳. از Select به صورت بازگشتی برای یافتن میانه‌ی جار $[n/5]$ میانه‌ی بدست آمده در مرحله دوم استفاده کرد.

۴. با استفاده از یک تفسیه تعبیریاتی Partition آرایه‌ی ورودی را حول میانه‌ها یعنی π تقسیم کرد. فرض کنید k تعداد عناصر در بخش پایین نقطه‌ی تقسیم باشد، پس $n - k$ تعداد عناصر بالای نقطه‌ی تقسیم خواهد بود.

۵. اگر $k \leq i$ از Select به صورت بازگشتی برای یافتن α مین عنصر کمینه در بخش پایینی استفاده کرد، و در غیر این صورت $(i - k)$ این عنصر کمینه را در بخش بزرگتر پیدا کند.

برای بررسی زمان اجرای Select، ما ابتدا یک حد بالین برای تعداد عناصر بزرگتر از عنصر تقسیم، π ، پیدا می‌کنیم. شکل ۱ برای تخمین این نتایج دهنده مفید است. حداقل تعداد میانه‌هایی که در مرحله دوم پیدا شده‌اند بزرگتر با مساوی میانه‌ی آنها یعنی π هستند. بنابراین حداقل $[n/5]$ گروه ۳ عنصر دارا هستند که بزرگتر از π هستند، مگر گروهی که کمتر از ۵ عنصر را دارد و دلیل اش هم بخش پذیر نبودن n به ۵ است و هم‌جنین گروهی که π در آن قرار دارد. بدون احتساب این دو گروه، توجه می‌کنیم که تعداد عناصر بزرگتر از π حداقل

$$2 \left(\left[\frac{1}{2} \left[\frac{n}{5} \right] \right] - 2 \right) \geq \frac{2n}{10} - 7$$

خواهد بود، به صورت مشابه تعداد عناصر کوچکتر از π حداقل $6 - n/10$ خواهد بود. بنابراین، در بدترین حالت، Select به صورت بازگشتی بزرگی حداقل $6 + 7n/10$ عنصر از مرحله ۵ اعمال خواهد شد.

ما می‌توانیم یک رابطه‌ی بازگشته برای زمان بدترین حالت اجرای `Select`, $T(n)$, بدست آوریم. مراحل ۱، ۲، و ۴ زمان $O(n)$ را مصرف می‌کنند. (مراحلی ۲ شامل $O(n)$ بار صدا کردن `insertion sort` بر روی اندازه‌ی (۱) است). مراحلی ۳ به اندازه‌ی $\lceil n/5 \rceil T(n/5)$ زمان مصرف می‌کنند، و مراحلی ۵ نیز حداقل $6 \lceil n/10 \rceil + 6$ زمان $T(\lceil n/10 \rceil + 6)$ مصرف می‌کنند، البته با فرض اینکه T بکرا و صعودی است. توجه کنید که برای $n > 20$ $n/10 + 6 < \lceil n/10 \rceil + 6$ و همچنین هر ورودی مرکب از A یا تعداد کمتری عنصر به زمان $O(1)$ نیاز دارد. بنابراین به رابطه‌ی بازگشته زیر می‌رسیم

$$T(n) \leq \begin{cases} \Theta(1) & \text{if } n \leq 8 \\ T(\lceil n/5 \rceil) + T(\lceil n/10 \rceil + 6) + O(n) & \text{if } n > 8 \end{cases}$$

حال ما با جای‌گذاری نشان دهیم که این زمان اجر خطی است. فرض کنید که برای یک c مفروض و هر $8 \leq n$ داشته باشیم $T(n) \leq cn$. با جای‌گذاری این فرض استقرا در سمت راست این رابطه بدست خواهیم آورد:

$$\begin{aligned} T(n) &\leq c\lceil n/5 \rceil + c(\lceil n/10 \rceil + 6) + O(n) \\ &\leq cn/5 + c + 7cn/10 + 6c + O(n) \\ &\leq 9cn/10 + 7c + O(n) \\ &\leq cn, \end{aligned}$$

زیرا ما می‌توانیم برای هر $n > 8$ را به قدر کافی بزرگ اختیار کنیم به طوری که $(n/10 - 7)c$ بزرگ‌تر از نابعی پاشد که با جمله‌ی $O(n)$ مشخص می‌شود. بنابراین زمان اجرای بدترین حالت `Select` خطی است. مانند مرتبسازی مقایسه‌ای (یعنی 9.1 را بینید)، `Select` و `Randomized-Select` اطلاعاتی راجع به مکان نسبی عناصر را فقط با مقایسه پیدا می‌کنند. بنابراین، ویژگی زمان خطی داشتن، نتیجه فرضیاتی راجع به ورودی تیست، درست مانند حالتی که در الگوریتم‌های مرتبسازی در فصل ۹ داشتیم. مرتبسازی در حالت مقایسه‌ای به زمان $O(\lg n)$ نیاز دارد، حتی در حالت متوسط (مسئله‌ی $1-9$ را بینید)، و بنابراین روش مرتبسازی و اندیس‌گذاری که در مقدمه‌ی این بخش آمده بود از نظر مرتبه ناکارآمد است.

تمرین‌ها

۱-۳ در الگوریتم `Select` عناصر ورودی را به گروه‌های ۵ تابی تقسیم کردیم. آیا الگوریتم بازهم در زمان خطی اجرا می‌شد اگر آن‌ها را به گروه‌های ۷ تابی تقسیم می‌کردیم؟ به گروه‌های ۳ تابی چه طور؟

۲-۳ الگوریتم `Select` را تحلیل کنید و نشان دهید که تعداد عناصر بزرگ‌تر از میانه‌ها بعنی $\frac{n}{2}$ ، و تعداد عناصر کوچک‌تر از $\frac{n}{2}$ حداقل $\lceil n/2 \rceil$ است، البته بشرط این که $n \geq 38$.

۳-۳ نشان دهید که چه گونه می‌توان کاری کرد که الگوریتم `quicksort` در بدترین حالت با $O(n \lg n)$ اجرا شود.

۴-۳ فرض کنید که یک الگوریتم فقط از مقایسه‌ها برای پاقن این عنصر کمینه در یک مجموعه از n عنصر استفاده می‌کند. نشان دهید که این الگوریتم می‌تواند ۱-۱ عنصر کوچک‌تر و ۱-۱ عنصر بزرگ‌تر را بزیبدون انجام مقایسه اضافی پیدا کند.

۶.۲ مرتب‌سازی خارجی

۵-۲

پک زیربرنامه برای پیدا کردن میانه که در بدترین حالت با زمان خطی عمل می‌کند به صورت پک «جمعیه میانه» داده شده است، پک الگوریتم خطی ساده را برای حل مستلزم انتخاب هر مرتبه‌ی آماری طراحی کرد.

۶-۳

k -امین quantile‌ها از پک مجموعه n عضوی، $1 - k$ -مرتبه‌ی آماری هستند که مجموعه‌ی مرتب شده را به k مجموعه با اندازه‌ی برابر تقسیم می‌کنند (یا اختلاف پک). پک الگوریتم با زمان $O(n \lg k)$ ارائه دهد که k -امین quantile‌های پک مجموعه را پیدا کند.

۷-۳

پک الگوریتم با زمان اجرای $O(n)$ پیدا کنید، به طوری که با گرفتن پک مجموعه S از n عضو متمایز و پک عدد مشت $i \leq k$, k -امین اعداد در S را که به میانه‌ی S نزدیک هستند را پایابد.

۸-۳

فرض کنید که $[1..n] = X$ و X دو آرایه باشند، که هر کدام حاوی n عدد مرتب شده هستند. پک الگوریتم با زمان اجرای $O(\lg n)$ طراحی کنید که بتواند میانه‌ی تمام $2n$ عنصر موجود در آرایه‌های X و Y را پیدا کند.

۹-۳

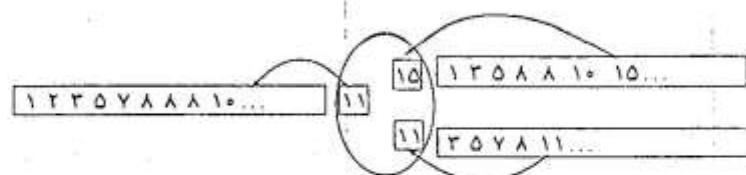
پروفسور $Olaf$ در حال همکاری با پک شرکت نفی است، که این شرکت در حال طراحی پک خط لوله‌ی عظیم است که از شرق به غرب در یک منطقه‌ی نفی با n چاه می‌گذرد. از هر چاه، پک خط لوله‌ی فرعی که مستقیماً وار نزدیکترین راه (شمال یا جنوب) به لوله‌ی اصلی وصل می‌شود و وجود دارد، مانند شکل ۱۰.۲. با در دست داشتن مختصدهای z و لوازمه‌ها، پروفسور محل بهینه‌ی لوله‌ی اصلی را چگونه باید تعیین کند (بعنی باید مجموع طول لوله‌های فرعی کمینه شود؟ نشان دهد که محل بهینه را می‌توان در زمان خطی پیدا کرد).

۶.۲ مرتب‌سازی خارجی

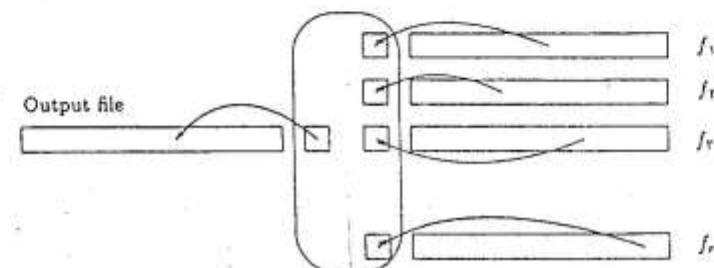
در این نوع مرتب‌سازی، تعداد دسترسی به عنوان معیار کارآئی الگوریتم در نظر گرفته می‌شود، چون زمان هر بار دسترسی به دیسک هزاران بار بیشتر از زمان دسترسی به حافظه‌ی اصلی است.

فرض می‌کنیم:

- اطلاعات بر روی فایل‌های به صورت ترتیبی ذخیره شده‌است. هر فایل شامل n رکورد است. هر رکورد پک کلید دارد.
- می‌خواهیم فایل‌ی سازیم که در آن رکوردها براساس کلیدهایشان مرتب باشند.
- با هر دسترسی به دیسک n رکورد خوانده می‌شود.
- تعداد فایل‌هایی که در پک زمان باز هستند m محدود است.
- تعداد حافظه‌ی اصلی فایل استفاده، مستقل از n است.
- عملیات مقایسه و محاسبات دیگر نقطه می‌تواند در حافظه‌ی اصلی انجام شود.



شکل ۱۱.۲: ادغام دو فایل مرتب.



شکل ۱۱.۲: ادغام دو فایل مرتب.

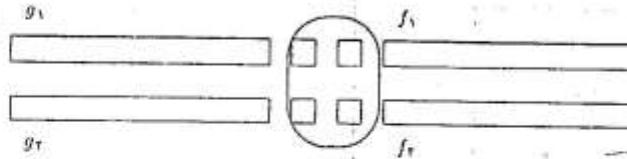
۱۱.۲ مرتب‌سازی خارجی مبتنی بر ادغام

این روش مبتنی بر ادغام دو قطعه‌ی مرتب از فایل برروی حافظه‌ی خارجی به مرتب با n_1 و m رکورد داده شده‌اند. می‌خواهیم ادغام شده‌ی این دو قطعه را به عنوان قطعه‌ای به طول $n_1 + m$ در حافظه‌ی خارجی پذیریم. این کار مشابهی عمل ادغام دو آرایه‌ی مرتب انجام می‌شود، در حافظه‌ی اصلی به دو میانگیر با اندازه‌ی بک رکورد نیاز داریم تا بتوان رکوردهای قطعه‌ها را در آن خواند. در اینداز هر قطعه بک رکورد می‌خوانیم و هر بار رکوردهای میانگیرها را با فرم مقابله‌ی می‌کنیم و رکورد کوچکتر را در انتهای قطعه خروجی می‌نویسیم و رکورد بعدی قطعه‌ای که رکورد را توشنده‌یم در همان میانگیر می‌خوانیم. این کار را تا خواندن توشنن همه‌ی رکوردهای دو قطعه‌ی ورودی ادامه می‌دهیم.

ادغام دو قطعه‌ی مرتب

اگر قطعه‌ی اول n_1 رکورد و قطعه‌ی دوم n_2 رکورد داشته باشد و $1 = k_1 + n_2 + n_1$ بار خواندن و همین تعداد بوشن می‌توان ادغام را انجام داد. این کار در حالت کلی با $\lceil \frac{n_1}{k_1} \rceil + \lceil \frac{n_2}{k_1} \rceil$ بار دسترسی به دیسک فایل انجام است. مقدار حافظه‌ی مورد نیاز در این حالت به اندازه‌ی $2k_1$ رکورد می‌باشد. (به شکل ۱۱.۲ توجه کنید).

۶.۲ مرتب‌سازی خارجی



شکل ۱۲.۲: الگوریتم MergeSort خارجی

ادغام ۲ قطعه‌ی مرتب

با همین روش می‌توانیم ۲ قطعه‌ی مرتب از فایل‌های مختلف را با هم ترکیب کنیم. (به شکل ۱۱.۲ نوچه کنید). تعداد دسترسی های برابر $\sum_{i=1}^{k-1} 2^i = 2^k - 2$ بار و مقدار حافظه‌ی مورد نیاز: بیانداره‌ی k رکورد.

الگوریتم کلی

برای $i = 1$ بیان می‌شود، چهار فایل، f_1, f_2, f_3 و f_4 احتیاج است.

۱. فایل ورودی را به دو فایل f_1 و f_2 با حداقل تعداد یک رکورد اختلاف تقسیم کن.

۲. برای $M = 1, \dots, M$ از مراحل زیر را تکرار کن:

در این مرحله فرض می‌کنیم که f_1 و f_2 (یا f_3 و f_4) شامل قطعه‌ای به طول 2^{M-1} هستند و هر قطعه مرتب است و تعداد قطعات دو فایل ورودی حداقل یک واحد اختلاف دارد.

(۲-۱) f_1 و f_2 را به صورت فایل‌های ورودی در نظر می‌گیریم. قطعات با شماره‌های پکان f_1 و f_2 را با پکدیگر ادغام کن و قطعه‌ای به طول دو برابر ایجاد کن. حاصل این ادغام قطعه‌ای مرتب به طول 2^M (جزء حداقل یک قطعه به طول کمتر) است این قطعات را بدتریب پکبار در ۱ و ۲ بار دیگر در f_3 و f_4 بنویس.

(۲-۲) f_3 و f_4 را به عنوان فایل‌های ورودی و f_1 و f_2 را به عنوان فایل‌های خروجی در نظر بگیر و مرحله‌ی بالا را تکرار کن.

- با استفاده از نوان نشان داد که در انتهای مرحله‌ی ۲ هر فایل خروجی دارای قطعه‌هایی مرتب و به طول 2^M است. بجز حداقل یک قطعه که طولش از 2^M کمتر است. همچنین تعداد قطعه‌های دو فایل خروجی حداقل یک واحد اختلاف دارند.

- بنابراین برای $M = \lceil \log n \rceil$ (تعداد تکرار حلقه) یکی از فایل‌های خروجی حاوی یک قطعه‌ی مرتب شامل تمام n رکورد فایل ورودی و دیگری خالی است.

- با توجه بداین که در هر تکرار همه n رکورد پکبار خوانده و یک بار نوشته می‌شوند، تعداد دسترسی به دیسک در مجموع برای $\lceil \log n \rceil + 1$ است ($2n$ بار خواندن و نوشتن برای تقسیم فایل اصلی).

- برای حالت $1 > k$, این تعداد برابر $(\lceil \frac{n}{k} \rceil \lceil \log n \rceil + 1) 2$ خواهد بود.
- با تقسیم فایل ورودی به 2 فایل با اندازه‌های بکسان و یا استفاده از 2 حافظه نیز می‌توان فایل را مرتب کرد در آن صورت، تعداد دسترسی به دیسک $2n \lceil \log_2 n \rceil + 2n$ می‌شود و در حالت کلی برابر $(\lceil \frac{n}{k} \rceil \lceil \log_2 n \rceil + 1) 2^r$.
- در حالت کلی به حافظه‌ای به اندازه‌ی $k(r+1)$ نیاز است.
- حالت کلی را Multiway Merge می‌گویند.

مثال:

```

f1 28 3 93 10 54 65 30 90 10 69 8 22
f2 31 5 96 40 85 9 39 13 8 77 10

g1 28 31 | 93 96 | 54 85 | 30 39 | 8 10 | 8 10
g2 3 5 | 10 40 | 9 65 | 13 90 | 69 77 | 22

f1 3 5 28 31 | 9 54 65 85 | 8 10 69 77
f2 10 40 93 06 | 13 30 39 90 | 8 10 22

g1 3 5 10 28 31 40 93 96 | 8 8 10 10 22 69 77
g2 9 13 30 39 54 65 85 90 |

f1 3 5 9 10 13 28 39 31 39 40 54 65 85 90 93 96
f2 8 8 10 10 22 69 77

g1 3 5 8 8 9 10 10 10 13 22 28 30 31 39 40 54 ...

```

۲.۱.۲ مرتبسازی خارجی Polyphase

- سه (در حالت کلی به $1 = r$) فایل لازم دارد.
- به فایل ورودی به کمترین تعداد رکورد با پیشترین کلید اضافه کن تا n برابر F_r (نامین عدد فیبوناچی) شود.
- فایل ورودی را به دو فایل با اندازه‌های F_{r-1} و F_{r-2} تقسیم کن ($F_r = F_{r-1} + F_{r-2}$) شود.
- بنداره‌ی $M = ?$ بار نکرار کن
- فرض کنید از سه فایل f_1 , f_2 دارای F_m قطعه‌ی مرتب (در ایندا $F_m = F_r$) است که اندازه‌ی هر قطعه F_r است (در ایندا $1 = F_r = F_{r-1} + F_{r-2}$).
- همین f_2 دارای F_{m-1} قطعه‌ی مرتب است که اندازه‌ی هر قطعه F_{r-1} است. و f_2 حالی است

7.2 مرتبسازی خارجی

۵۵

قطعدی مرتب از فایل ۱ را با همین تعداد قطعدی مرتب از فایل ۲ را با هم ادغام کن و حاصل را در فایل ۳ بنویس.

حالا f_1 خالی، f_2 دارای $F_m = F_{m+1} - F_{m-1}$ قطعه به اندازه $f_{r+1} - f_r$ و f_3 دارای F_m قطعدی مرتب به اندازه $F_r + F_{r+1} = F_r + F_{r+1}$ است.

نامگذاری فایل‌ها را به تناسب تغییر بدء.

مثال: $n = 24$

after pass	f_1	f_2	f_3
initial	13(1)	21(1)	-
1	-	8(1)	13(2)
2	8(3)	-	5(2)
3	3(3)	5(5)	-
4	-	2(5)	3(8)
5	2(13)	-	1(8)
6	1(13)	1(21)	-
7	-	-	1(34)

فصل ۳

داده‌ساختارها

۱.۳ دسته‌بندی

۱) داده‌ساختارهای ساده

- » لیست‌ها (لیست‌های ساده‌ی یک طرفه، دو طرفه، دوار)
 - » پسته‌ها
 - » صفحه‌ها
 - » درخت‌ها

۲) داده‌گونه‌ی انتزاعی مجموعه‌ها

» مرھنگ داده‌ای (درج، حذف و جست‌و‌جو)

• جدول پراکندگی (hash table): باز، پسته، متوسط $O(1)$

• درخت دودویی جست‌و‌جو (BST): حداکثر $O(n)$ و متوسط $O(\lg n)$

• درخت دودویی جست‌و‌جوی با ارتفاع $O(\lg \lg n)$

• درخت دودویی جست‌و‌جوی متوازن

(a) درخت AVL (درختی که اختلاف ارتفاع در زیردرخت هر گرهی آن حداکثر ۱ باشد)

(b) درخت اقرمز-سباه (Red-Black tree)

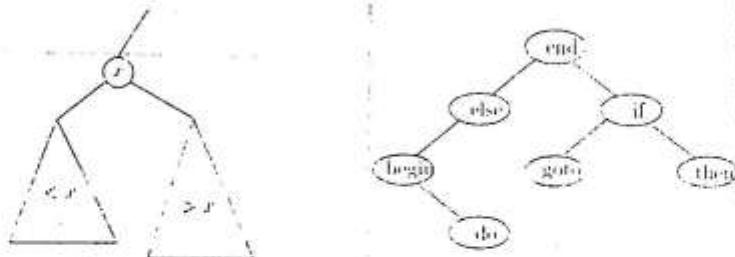
• درخت‌های کاملاً متوازن (ارتفاع $O(\lg n)$)

(a) درخت ۲-۳

(b) درخت (ابی)

• صف اولیت (درج و حذف کوچکترین عنصر): $O(\lg n)$

Disjoint Find-Merge (پیداکردن، ادغام)



شکل ۳.۲: درخت دودویی جستجو و بک متال.

۳.۲ داده‌ساختارها برای فرهنگ داده‌ای

۳.۲.۱ درخت دودویی جستجو

تعریف: درختی دودویی که کلیه‌ی عناصر زیر درخت چپ گردد، رکنتر از ۰ و کلیه‌ی عناصر زیر درخت راست آن بیش نزدیک باشد. (شکل ۳.۲)

سمانیسین ترتیب بک درخت دودویی جستجو و حوزه مرتب‌شده‌ی عناصر را تولید می‌کند.

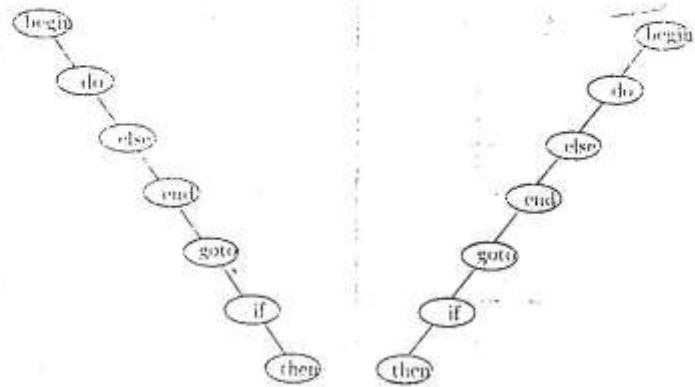
درست برج در ارتفاع درجت حاصل بسیار مذکور است (شکل‌های ۳.۲ و ۳.۳)

$$\text{ارتفاع} \leq n - 1 \quad (\text{چرا}) \\ O(\lg n)$$

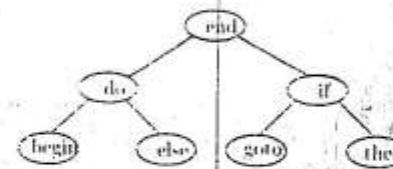
عداد درخت‌های دودویی جستجو که با $n < n_1 < \dots < n_k = n$ می‌توان ساخت:

$$T(n) = \sum_{i=1}^n T(i-1)T(n-i), T(0) = 1 \\ = \frac{1}{n+1} \binom{n}{n}$$

عدد کاملاً



شکل ۲.۳: درخت دودویی جست و جو با پین ارتفاع.



شکل ۲.۴: درخت دودویی جست و جو با کمترین ارتفاع.

```
FUNCTION MEMBER (x: LabelType; A: BST): BOOLEAN;
begin
  if A = nil then return (false);
  if A^.label = x then return (true);
  if x < A^.label then
    return (member(x, A^.left))
  else return (member(x, A^.right))
```

```
PROCEDURE INSERT (x: LabelType; var A: BST);
begin
  if A = nil then begin
    new(A); A^.label := x;
    A^.left := nil;
    A^.right := nil;
  end else if x < A^.label then
    insert (x, A^.left);
  else if x > A^.label then
    insert (x, A^.right)
  end;
```

حذف کوچکترین عنصر

```

FUNCTION DELETENIN (var A:BST ): LabelType;
var t: BST;
begin
  if A = nil then error ('no such element');
  if A^.left = nil then begin
    deletemin := A^.label;
    t := A; A := A^.right;
    dispose(t);
  end
  else return (deletemin(A^.left));
end;

```

حذف

```

PROCEDURE DELETE (x:labeltype; var A:BST);
var t: BST;
begin
  if a = nil then error ('tree is empty');
  if x < A^.label then delete(x,A^.left)
  else if x > A^.label then delete(x,A^.right)
  else begin (A^.label=x)
    t:=A;
    if A^.left = nil then begin
      A:=A^.right;
      dispose (t);
    end
    else if A^.right = nil then begin
      A:=A^.left;
      dispose (t);
      else A^.label:=deletemin(A^.right);
    end
  end;
end;

```

۲.۲.۴ متوسط ارتفاع درخت دودویی جستجو

اگر $a_0 < a_1 < \dots < a_n$ به صورت نعادی و با احتمال بگسان وارد یک درخت دودویی جستجوی تبی T شوند، انساب می‌کنم که متوسط ارتفاع T برای $O(\log n)$ است.

- فرض کند، اولین عنصری است که درج می‌شود.

• ریشه‌ی درخت خواهد بود.

- ناتای n_{i-1} به هر ترتیبی که درج شوند در زمره‌رخت چپ فرار خواهد گرفت.

• ناتای n_{i-1} در زمره‌رخت راست خواهد بود.

- اگر $h(n)$ متوسط ارتفاع T باشد، داریم:

$$h(n) = \lambda + \sum_{i=1}^n \frac{1}{n} \max\{h(i-\lambda), h(n-i)\}$$

فرض می‌کنیم $h(n) \leq c \log n$ برای یک $c > 0$.

با توجه به این که $h(i)$ یکتابع غیرنژولی است.

$$\begin{aligned} h(n) &= \lambda + \frac{1}{n} \left(\sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} h(n-i) + \sum_{i=\lceil \frac{n}{2} \rceil + 1}^n h(i-\lambda) \right) \\ &\leq \lambda + \frac{c}{n} \sum_{i=\lceil \frac{n}{2} \rceil + 1}^n h(i-\lambda) \\ &\leq \lambda + \frac{c}{n} \sum_{i=\lceil \frac{n}{2} \rceil + 1}^n i \\ &\leq \lambda + \frac{c}{n} \sum_{i=\lceil \frac{n}{2} \rceil + 1}^n c \log i \\ &\leq \lambda + \frac{c}{n} \log \frac{(n-\lambda)!}{(\frac{n}{2}-\lambda)!} \\ &\leq \lambda + \frac{c}{n} c! \log n! - \log \left(\frac{n}{2} \right)^{\frac{n}{2}} \end{aligned}$$

با توجه به این که $k \cdot n \log n \leq \log n!$ (معنی $\log n! = \Theta(n \log n)$)

$$\begin{aligned} h(n) &\leq \lambda + \frac{c}{n} (k \cdot n \log n - k \cdot \frac{n}{2} \log \frac{n}{2}) \\ &\leq \lambda + ck(k - \frac{k}{2} \log \frac{n}{2} - \frac{k}{2}) \end{aligned}$$

می‌توان $k_1 < k$ را پیدا کرد که $\lambda - ck_1 < \lambda/2 - ck$ پس

$$h(n) \leq c \log n$$

۲.۳ صف اولویت (Priority Queue)

۲.۳ صف اولویت (Priority Queue)

صف اولویت یک داده‌ساختار است برای نگهداری عناصر یک مجموعه مکاری رود به طوری که هر یک از عناصر یک کلید دارد که نمایانگر اولویت آن عصر می‌باشد. اعمال مینیمی که بر روی یک صف اولویت انجام می‌شوند عبارتند از:

- * `Insert(Q, x)`: عنصر x را در صف Q وارد می‌کند.
- * `Maximum(Q)`: عنصر با اولویت بیشتر را برمی‌گرداند.
- * `Extract-Max(Q)`: عنصر با اولویت بیشتر را از صف حذف و برمی‌گرداند.

با توجه به شباهت میان `heap` و صف اولویت می‌توان اعمال فوق را با ساختار `heap` پیاده‌سازی کرد. در این رابطه روابه‌های زیر پیاده‌سازی می‌شوند:

- * `Heap-Insert (A, key)`: در این روابه یک عنصر به Heap اضافه می‌شود. برای انجام این کار ابتدا یک مرگ جدید به درخت (در جای لارم) با کلید `key` اضافه می‌شود و سپس `key` قرار گرفته در مکان مناسب به سمت بالا حرکت می‌کند.

```

    Heap_Insert(A, key)
1      Heap_size(A) <- Heap_size(A)+1
2      while(i>1 and A[parent(i)]<key)
3          do   A[i] <- A[parent(i)]
4                  i <- parent(i)
5      A[i] <- key

```

زمان اجرا: از آن‌جاکه طول مسیر پیمایش شده حداقل ارتفاع است. $T(n) = O(\log n)$.

- * `Heap-Maximum (A)`: عنصر با اولویت بیشتر را که در $A[1]$ قرار دارد برمی‌گرداند.

- * `Heap-Extract-Max (A)`: عنصر با اولویت بیشتر را از `heap` حذف کرده و متدار آن را برمی‌گرداند.

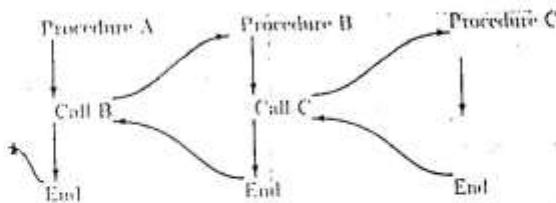
```

    Heap_Extract_Max(A)
1      if heap_size(A)<1
2          Then error "Heap underflow"
3      max <- A[1]
4      A[1] <- A[heap_size(A)]
5      heap_size(A) <- heap_size(A)-1
6      Heapify(3,1)
7      return max

```

واضح است که زمان اجرای این روش $T(n) = O(\log n)$ است. در نتیجه می‌توان بک صف اولویت را با استفاده از heap پاده‌سازی کرد و هرینه‌ی اعمال فوق حداقل $O(\log n)$ خواهد بود.

۴.۲ تبدیل الگوریتم‌های بازگشتی به غیربازگشتی



شکل ۴.۲: انتقال کنترل برنامه در فراخوانی و بازگشت

۴.۳ تبدیل الگوریتم‌های بازگشتی به غیربازگشتی

الگوریتم‌های بازگشتی شامل دو مرحله‌ی مهم هستند:

۱. عمل فراخوانی

۲. بازگشت از یک فراخوانی

هدف آن است که بتوانیم دو مرحله‌ی فوق را شبیه سازی نماییم. فراخوانی مانند وقته عمل می‌کند. کلیدی متغیرها در لحظه فراخوانی ذخیره می‌شوند، عملیاتی طورت می‌گیرد و هنگام بازگشت وضعیت پیش از فراخوانی مجددًا بازسازی می‌شود.

در نقطه H مقادیر متغیرها (به جز متغیرهای از نوع آدرسی Called By Reference) همان مقدار قبل از فراخوانی را دارند. فراخوانی می‌تواند به رویه‌های دیگر و با به خود همان رویه (در برنامه‌های بازگشتی) باشد. هر فراخوانی ممکن است نتیجه‌هایی به همراه داشته باشد ولی مقدار متغیرها تغییر نمی‌کند.

هر فراخوانی (Call) شامل مراحل زیر است :

۱. کلیدی متغیرهای محلی (در حالت کلی کلمه متغیرهای دسترسی‌مند) و مقدارهایان در بسته‌ی سیم فرار می‌گیرند (Push).

۲. آدرس بازگشت به بسته متنقل می‌شود (Push).

۳. عمل انتقال پارامترها (Parameter Passing) صورت می‌گیرد. پارامترها ممکن است از نوع آدرسی (Address) یا آدرسی (Variable) باشد.

۴. کنترل برنامه (شات شمارنده‌ی برنامه Program Counter) به اندیای رویه‌ی جدید اشاره می‌کند. عمل بازگشت (Return) عکس عملیات فوق را انجام میدهد.

۱. مقدارهای متغیرهای محلی را از رکورد بالای بسته برداشته و در خودشان فرار می‌دهم.

۲. آدرس بازگشت را از بالای بسته به دست می‌آورم.

۳. آخرین رکورد را از بسته برگمی دارم (Pop).

۴. کنترل برنامه را از آدرس بارگذشت (بند ۲) ادامه می‌دهیم.

برای روشن شدن این مطلب به مثال رمر نوچه می‌کیم. در مساله‌ی مرچ‌های هانوی، هدف انتقال «سکه‌ی سرواح‌دار از میله‌ی (پاسخ) مبدأ به میله‌ی مقصد با کمک میله‌ی سوم است. در انتقال سکه‌ها دونکنه باید رعایت شود:

۱. هر بار بالاترین سکه باید حرکت داده شود.

۲. سکه‌ی برزگر بر روی سکه‌ی کوچکتر فرار نگیرد.

خروجی برنامه گام‌های انتقال سکه‌ها از میله‌ی مبدأ به میله‌ی مقصد است. برای حل مساله میله‌ی مبدأ را *from* و میله‌ی مقصد را *to* و میله‌ی کمکی را *h* (help) می‌نامیم. برنامه‌ی انتقال سکه‌ها به طور بارگذشتی به صورت زیر است:

```
procedure Hanoi(n,f,t,h:Integer);
begin
  if (n = 1) Then
    writeln(f,'->',t)
  else begin
    Hanoi(n-1,f,h,t);
    A ->
    writeln(f,'->',t);
    B ->
    hanoi(n-1,h,t,f)
  end
end;
```

مرحله‌ی ۱: در این مرحله یک فراخواست انجام می‌شود، ارزش متغیرها ($h=2$, $t=3$, $f=1$, $n=3$) و آدرس محل فراخواست (Ret_Addr='A') در پسته ذخیره شده و متغیرها ارزش جدیدی پیدا می‌کنند. (شکل ۳.۵.۲)

مرحله‌ی ۲: یک فراخواست مستقل دیگر صورت می‌گیرد، و ارزش متغیرها و آدرس محل فراخواست ('A') در پسته فرار می‌گیرد. مقادیر متغیرها عبارت است از:

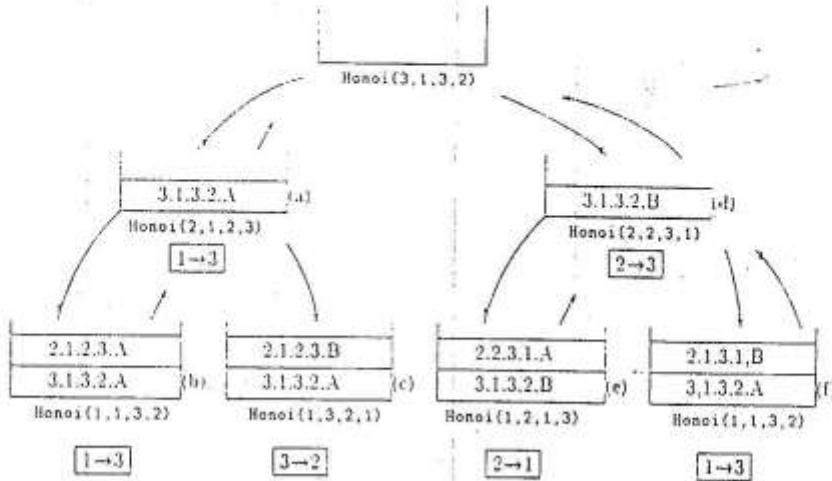
$n = 1$, $f = 1$, $t = 3$, $h = 2$

و یک پرش به آنشای برنامه انجام می‌شود. (شکل ۳.۵.۲)

مرحله‌ی ۳: جون $1 * n$ است قسمت اول برنامه اجرا می‌شود. (خروجی ۳ → ۱)

مرحله‌ی ۴: یک بارگذشت (Return) صورت می‌گیرد. مقادیر متغیرهای محلی و آدرس بارگذشت از پسته برداشته می‌شوند یعنی داریم:

$n = 2$, $f = 1$, $t = 2$, $h = 3$, Ret_Addr = 'A'



شکل ۳: مراحل اجرای فرآخوانی Hanoi(3,1,3,2)

آخرین رکوره پنجه را (Pop) می کیم و کنترل برنامه را از آدرس بازگشت ('A') با متدارهای جدید متغیرها ادامه می دهم. شکل ۵.۲ (۱) خروجی ۲ → (۱)

مرحله ۵: یک فرآخوانی مستقل دیگر انجام می شود. متدارهای متغیرها و آدرس بازگشت (Ret_Addr='B') در پنجه ذخیره شده و پس از انجام عمل انتقال پارامترها داریم (شکل ۵.۲ (۱))

$n = 1, f = 3, h = 2, t = 1$

مرحله ۶: ۱ * n پس فرآخوانی صورت نمی گیرد. (خروجی ۲ → (۳))

مرحله ۷: عمل بازگشت پس از طی قدم های زیر صورت می گیرد:

۱. ارزش متغیرها را از مالایی پنجه برداشته و داریم:

$n = 2, f = 1, t = 2, h = 3$

۲. آدرس محل فرآخوانی ('B') را از پنجه بر می داریم.

۳. پنجه را (Pop) می کیم.

۴. برنامه را از ('B') با ارزش های جدید متغیرها دیال می کنم. (شکل ۵.۲ (۱))

مرحله ۸: یک بازگشت دیگر صورت می گیرد و داریم:

$n = 2, f = 1, t = 3, h = 2, Ret_Addr = 'A'$

برنامه را از ('A') بی می گیریم. (خروجی ۳ → (۱))

فصل ۳ داده‌ساختارها

۶۸

مرحله‌ی ۹: پس از اجام عمل فراخوانی در پسته یک رکورد جدید با مقادیر $n = 3, f = 1, t = 3, h = 2, \text{Ret_Addr} = 'A'$ داریم و ارزش جدید متغیرها عبارتند از (شکل ۰.۳ (iii)):

$n = 2, f = 2, t = 3, h = 1$

مرحله‌ی ۱۰: یک فراخوانی انجام می‌گیرد و ارزش جدید متغیرها عبارتند از: $n = 1, f = 2, t = 1, h = 3$ (شکل ۰.۳ (iv))

مرحله‌ی ۱۱: چون $n = 1$ فراخوانی دیگری در این مرحله روی نمی‌دهد. (خروجی ۱ $\rightarrow 2$)

مرحله‌ی ۱۲: یک بازگشت صورت می‌گیرد. و متدارهای جدید متغیرها به صورت زیر است:

$n = 3, f = 2, t = 3, h = 1, \text{Ret_Addr} = 'A'$

برنامه را از ('A') دسال می‌کیم - (شکل ۰.۳ (v)). خروجی ۳ $\rightarrow 1$

مرحله‌ی ۱۳ فراخوانی مستقل داریم. متدارهای متغیرها و آدرس محل فراخوانی ($\text{Ret_Addr} = 'B'$) را در پسته وارد می‌کیم و ارزش جدیدی به متغیرها می‌دهیم (شکل ۰.۳ (vi)). (شکل ۰.۳ (f))

مرحله‌ی ۱۴: $n = 0$ فراخوانی دیگری صورت نمی‌گیرد. (خروجی ۳ $\rightarrow 1$)

مرحله‌ی ۱۵: در این مرحله یک بازگشت رخ می‌دهد (شکل ۰.۳ (vii)).

$n = 2, f = 2, t = 3, h = 1$

مرحله‌ی ۱۶: یک بازگشت داریم.

$n = 3, f = 1, t = 3, h = 2$

برنامه را از ('B') دسال می‌کیم.

مرحله‌ی ۱۷: سرانه احری روبه و بازگشت به برنامه‌ی اصلی

۰.۳ برنامه‌ی غیر بازگشتی

رویه‌ی سرج شای غایبی: صورت غیر بازگشتی (Non-Recursive) نامیده شود. بر این است:

۶۸

```

Const Max = 100;
Type Stack = ...;

Procedure NR_Hanoi(n,f,t,h:integer);
Var
  s : Stack;
  Ret_Addr : char;
  Label 1,99;
Begin
  MakeNull(s);
  1: If (n = 1) Then Begin
    Writeln(f,'->',t);
    Goto 99;
  End
  Else Begin
    Push(s,n,f,t,h,'A');
    n := n-1;
    Swap(h,t);
    Goto 1;
  End;
  99: If Empty(s) Then Exit;
  Ret_Addr, h, t, f, n <- Top(s);
  Pop(s);
  Case Ret_Addr Of
    'A': Begin
      Writeln(f,'->',t);
      Push(s,n,f,t,h,'B');
      n := n-1;
      Swap(h,f);
      Goto 1;
    End;
    'B': Goto 99;
  End;
End;

```

در این برنامه یک متغیر با گونه‌ی (Stack) برای شبیه‌سازی پشته تعریف شده است. آدرس بارگشت فراخوانی از گونه‌ی نویسه است. Label برای مشخص نمودن محل بروش هاست.

ابتدا برنامه‌ی پشته (5) را برای تشخیص ختم بارگشت نهی می‌کنیم (MakeNull). اگر (n=1) یک بروش به انتهای برنامه صورت می‌گیرد (شبیه سازی عمل (Return)) و در غیر این صورت اولین فراخوانی مانند شبیه‌سازی مود: به این ترتیب که ابتدا کلیه‌ی متغیرهای محلی (s,n,f,t,h) و آدرس بارگشت ('A') از پشته فرار می‌گیرد، پس عمل انتقال بارگذاری انجام می‌شود و سرانجام یک بروش به اول برنامه صورت می‌گیرد. اگر پشته فرار می‌گیرد، یک بارگشت به برنامه‌ی اصلی صورت می‌گیرد و در غیر این صورت متغیرهای محلی و آدرس بارگشت مقدار قابل وجود را پیدا می‌کنند و بالاترین رکورد پشته برداشته (Pop) می‌شود. پسنه به مقدار آدرس بارگشت دو کار بر صورت می‌گیرد:

۱. فراخوانی دوم شبیه سازی می شود.

۲. پرش انجام می شود.

برنامه ای بالا نمونه ای از یک تبدیل برنامه بازگشته به غیر بازگشته است، در این برنامه اصول برنامه نویسی چندان رعایت نشده ولی می توان آن را بهینه نمود.

۱.۵.۲ حذف آخرین بازگشت (Tail Recursion)

آخرین فراخوانی بازگشته که بعد از آن در هیچ شرایطی دستوری که از متغیرهای متغیرها استفاده کند، اجرا نشود را آخرین بازگشت میگوییم. این بازگشت را می توان بدون استفاده از پیشنه حذف کرد. برنامه زیر نمونه ای از آخرین بازگشت است.

```
Procedure A
Begin
  ...
  Call A
  x ->
End;
```

در بازگشت به این فراخوانی (A) متغیرهای محلی متدارهایستان تغییر می کند و اجرای برنامه از نقطه ای (x) دیگر می شود. ولی (x) تنها یک بازگشت است. پنایین می توان عمل وارد کردن ارزش متغیرها و آدرس فراخوانی در پیشنه (Push) و حذف از پیشنه (Pop) را از برنامه حذف نمود و فراخوانی تنها به یک انتقال پارامتر و یک پرش به اول برنامه تبدیل شود. به عنوان مثال برنامه های هانوی را می توان به صورت زیر نوشت:

```
Procedure Hanoi(n,f,t,h:Integer);
Label 1;
Begin
  1: If (n = 1) Then WriteLn(f,'->',t)
    Else Begin
      Hanoi(n-1,f,h,t);
      WriteLn(f,'->',t);
      n := n-1; {parameter passing}
      Swap(f,h); {parameter passing}
      GoTo 1
    End
  End;
```

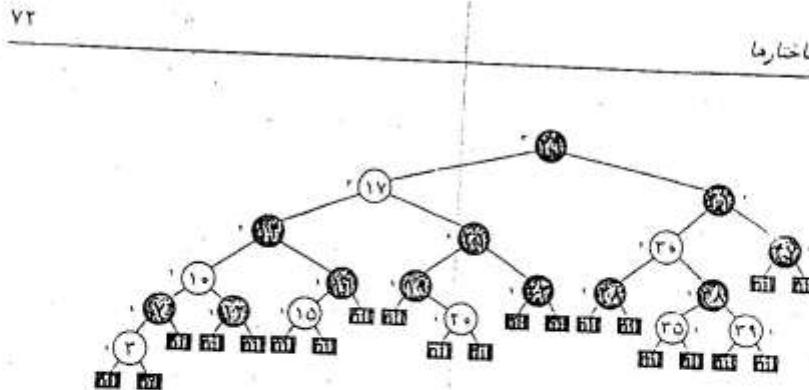
۵.۳ برنامه‌ی غیر بازگشته

اگر برنامه‌ی فوق به غیر بازگشته تبدیل شود، چون فقط یک آدرس بازگشت داریم، دیگر نیازی به نگذاری آدرس بازگشت نیست. روشی فوق به صورت زیر غیر بازگشته می‌شود:

```
procedure NR_Hanoi2(n,f,t,h:Integer);
Var s : Stack;
Label 1,99;

Begin
  MakeNull(s);
1: If (n = 1) Then Begin
    WriteLn(f,'->',t);
    Goto 99
  End Else
  Begin
    Push(s,n,f,t,h)
    n := n-1
    Swap(h,t);
    Goto 1
  End;
99: If Empty(s) Then Exit;
  n,f,t,h <- Top(s);
  Pop(s);
  WriteLn(f,'->',t);
  n := n-1;
  Swap(h,f);
  Goto 1;
End;
```

در برنامه‌ی برج های هانوی همه‌ی متغیرها از نوع ارزشی (Value) بودند. اما اگر متغیرها از نوع (Variable) باشد چه باید کرد؟ در این صورت می‌توانیم آدرس این متغیرها را به جای مقادیرشان در پسته وارد کیم.



شکل ۶.۲: مثال درخت قرمز- سیاه

۶.۳ درخت قرمز- سیاه (Red-Black Tree)

هدف: ایجاد درخت جستجوی دودویی که موارد زیر را تضمین نماید:

۱. $O(\lg n)$ که n تعداد گرهای h ارتفاع درخت می‌باشد. واضح است که در اینصورت عمل جستجوی پک عنصر در فرهنگ داده‌ای «در بدترین حالت» $O(\lg n)$ خواهد بود.

۲. توانمندی عملیات درج و حذف در فرهنگ داده‌ای را با الگوریتمی با پیچیدگی $O(\lg n)$ انجام دهیم بگونه‌ای که خاصیت ۱ حفظ شود.

برای رسیدن به اهداف فوق درخت قرمز- سیاه را تعریف می‌کیم:

تعریف: درخت قرمز- سیاه، درخت جستجویی می‌باشد که هر عنصر آن یک مؤلفه‌ی رنگ (color) دارد که می‌تواند سیاه یا قرمز باشد. اشاره‌گرگاری nil در این درخت را برگهای nil و سایر عناصر شامل برگهای درخت اولیه را گره‌های داخلی (Internal Nodes) می‌نامیم. نمونه‌ای از درخت قرمز- سیاه در «شکل ۶.۳» آورده شده است.

۱.۶.۲ خواص درخت قرمز- سیاه:

۱. هر گره با قرمز است و با سیاه.
۲. هر برگ nil سیاه است.
۳. دو فرزند یک گره قرمز سیاهند (پدر یک گره قرمز نمی‌تواند قرمز باشد).
۴. هر سیبر ساده از یک گره به برگ فرزند (نه لزوماً فرزند مستقیم) شامل تعداد پکسایی گره سیاه می‌باشد.
۵. ریشه درخت سیاه است (این شرط، از شرط اساسی نیست).

تعریف و تضیییان اولیه:

۱. سیاه ارتفاع گره x : بنایه تعریف $(x)bh$ ^۱ برابر است با تعداد گرهای سیاه از ریشه یک برج فرزند.

۲. عنوی گره x :

```
if parent[x] = right[parent[parent[x]]]
  then uncle[x] := left[parent[parent[x]]]
else uncle[x] := right[parent[parent[x]]]
```

قضیه ۱ حداقل ارتفاع یک درخت قرمز- سیاه که دارای « گره داخلی » می باشد. برای $1 \leq h \leq \lg(n+1)$ است. برای اثبات قضیه فوق ایندا نشان می دهیم که یک زیردرخت به ریشه دلخواه ۰ حداقل دارای $1 - 2^{bh(x)}$ گره داخلی می باشد. برای اثبات از استقراء بر روی ارتفاع گره داخلی x در درخت استفاده می نماییم.

پایه: اگر ارتفاع x صفر باشد، x حتماً برج و در نتیجه nil خواهد بود. بنابراین زیردرخت به ریشه x حداقل $1 - 2^{bh(x)} = 1 - 2^0 = 1$ گره داخلی دارد (بدینه).

گام استقراتی: گره داخلی x را در نظر بگیرید. سیاه- ارتفاع x عددی مشخص می باشد و x دارای دو فرزند می باشد. هر کدام از فرزندان بر حسب اینکه قرمز باشد یا سیاه، سیاه- ارتفاعی برای $(x)bh(y)$ خواهد داشت. به علت اینکه هر فرزند x ارتفاعی کمتر از خود طبقه دارد می توانیم با استقراء از فرض استقراء نتیجه نگیریم که هر زیردرخت به ریشه یک فرزند x حداقل $1 - 2^{bh(x)}$ گره داخلی دارد. بنابراین زیردرخت به ریشه x حداقل $1 - 2^{bh(x)} = 1 - 2^{bh(x-1)} + 1 = 1 + 1 = 2$ گره داخلی خواهد داشت.

از طرف دیگر می دانیم که در درخت به ارتفاع h حداقل نیمی از گرهها (بدون درنظر گرفتن ریشه) بر روی هر مسیر ساده از ریشه به برج، سیاه هستند. زیرا در غیر اینصورت خاصیت ۲ از خواص درخت قرمز- سیاه نقض خواهد شد. نتیجتاً سیاه- ارتفاع ریشه درخت حداقل $\frac{1}{2} n$ خواهد بود. بنابراین:

$$n \geq 2^{\frac{1}{2}h} \Rightarrow h \leq \lg(n+1)$$

نتیجه: ارتفاع درخت قرمز- سیاه $O(\lg n)$ است.

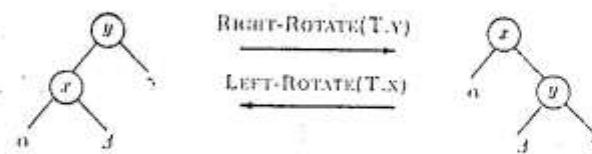
۶.۱.۳ دوران

برای بازیابی خواص درخت قرمز- سیاه بعد از عملیات درج و حذف در بعضی شرایط محصور خواهیم کرد که ریگ بعضی از گرهها را عوض کنیم با تغییراتی در ساختار اشاره گرهای اعمال نمائیم. به همین منظور دو عمل دوران (راستگرد و چیگرد) تعریف می نماییم. با توجه به «شکل ۶.۳» واضح است که این دو عمل هیچ اشکالی در خواص یک درخت دو دوستی ایجاد نخواهد نمود. شکل ۶.۳ مثالی از LEFT-ROTATE (T,x) را نشان می دهد.

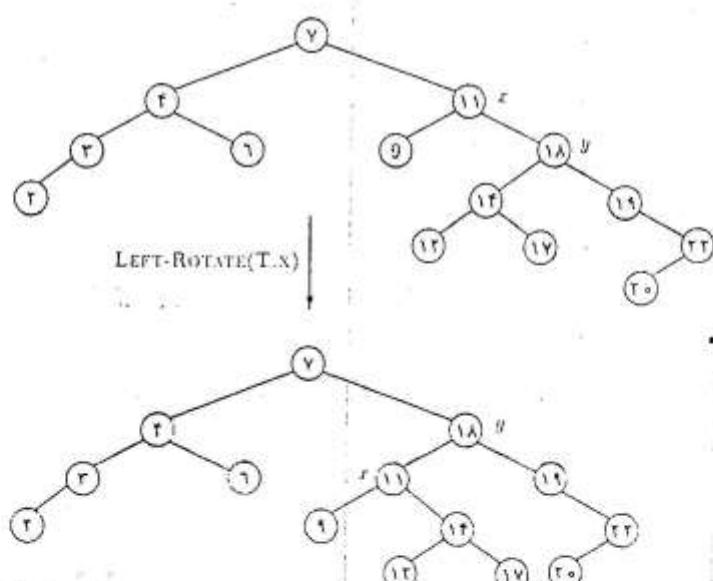
نکته: وقتی ما یک دوران چیگرد انجام می دهیم فرض می کنیم که فرزند راست گره مورد نظر nil نباشد.

^۱ Black Height

^۱ ریگ x بنایر است زیرا x را نمی شماریم.



شکل ۷.۲: دوران راستگرد و چیگرد



شکل ۸.۲: مثالی از $\text{LEFT-ROTATE}(T,x)$

۳.۳ درخت قرمز- سیاه (RED-BLACK TREE)

شبیه دستورات لازم برای پیاده‌سازی دوران چیزگرد در اینجا آورده شده است. دستورات لازم برای دوران راستگرد زیر شبیه به حالت چیزگرد می‌باشد. واضح است که در عمل فوق (۱) می‌باشد زیرا فقط اشاره‌گرها در اتریک دوران تغییر می‌کنند و سایر اجزا بدون تغییر می‌مانند.

```

Left-Rotate(T,x)
y <- right[x]
right[x] <- left[y]
if left[y] <> NIL then p[left[y]] <- x
p[y] <- p[x]
if p[x] = NIL
then Root[T] <- y
else if x = left[p[x]]
then left[p[x]] <- y
else right[p[x]] <- y
left[y] <- x
p[x] <- y

```

شبیه کد مربوط به

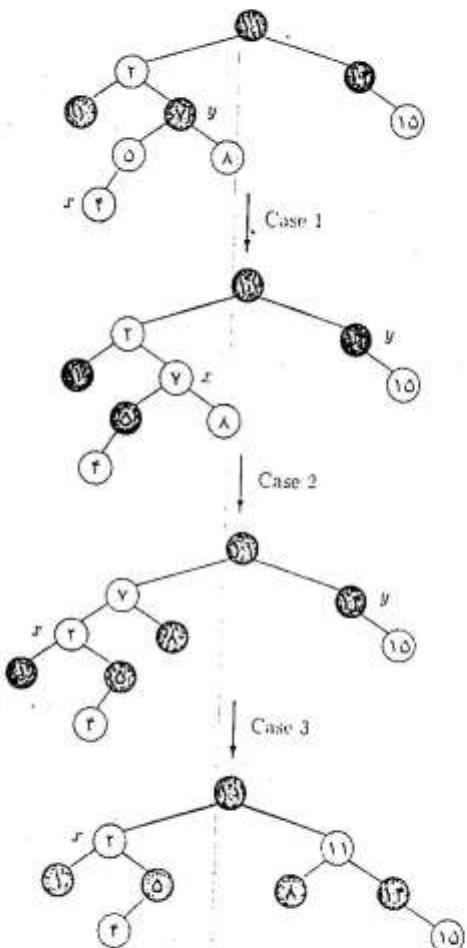
۳.۶.۳ درج

۱. براساس درج در درخت دودویی جستجو، \hat{x} را درج می‌نماییم.
۲. \hat{x} را قرمز می‌کیم. بدینهی است که سیاه-ارتفاع‌ها برای کلبه‌گره‌ها نایت می‌ماند.
۳. اگر پدر \hat{x} سیاه باشد درج به پایان رسیده است.
۴. اگر پدر \hat{x} قرمز باشد به سراغ عمومی \hat{x} بنام \hat{z} می‌رویم. (همانطور که در شبیه دستورات مربوط به RB-Insert دیده می‌شود برحسب اینکه Parent[x] فرزند چپ یا فرزند راست حد \hat{x} باشد، عمومی \hat{z} به ترتیب فرزند راست یا چپ حد \hat{x} خواهد بود)

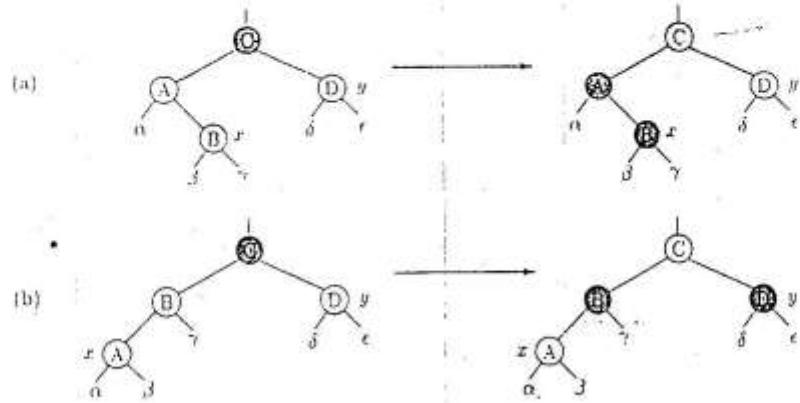
در قسمت بعد اینه اصلی این است که در هر مرحله با حفظ خاصیت ۴ مربوط به درخت قرمز- سیاه، لشکال موجود که مربوط به عدم برقراری خاصیت ۳ می‌باشد را برای ارتفاع فعلی درخت حل کنیم و مشکل را احیاناً به ارتفاعات بالاتر ارسال نمائیم و این کار را برای سطوح بالاتر آشنازد ادامه دهیم تا به ریشه بررسیم با شرط سوم (سیاه بودن پدر \hat{x}) برقرار شود و اجراء خاتمه پذیرد. واضح است که درجت صرفه زمانی پیار به تصحیح دارد اگه پدر \hat{x} قرمز باشد.

برحسب اینکه Parent[x] فرزند چپ یا راست حد \hat{x} باشد، ۶ حالت متفاوت پیش می‌آید که ۳ حالت دیگر عنانه متفاوت با سه حالت زیر می‌باشد. این حالات بر اساس رنگ عمومی \hat{z} یعنی لاعین می‌شوند.

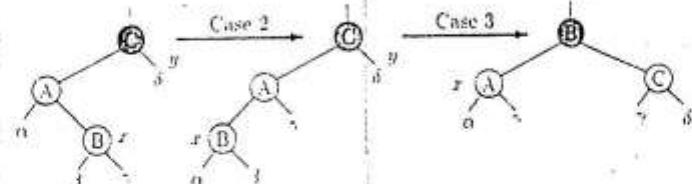
حالات اول: لا قرمزی است.



شکل ۴.۲: مثالی از درج



شکل ۱۰.۳: حالت اول برای RB-INSERT



شکل ۱۱.۳: حالت های دوم و سوم برای RB-INSERT

فصل ۳ داده‌ساختارها

حالت دوم: سیاه است و فرزند راست پدرش است.

نکته: دیده می‌شود که در بایان اجرای RB-Insert برای حفظ خاصیت ۵، ریشه درخت سیاه می‌شود. شکل‌های ۱۰.۲ و ۱۱.۳ حالات‌های مختلف درج را نشان می‌دهند.

```
RB-Insert(T,x)
1. Tree-Insert(T,x)
2. color[x] <- Red
3. while x <> root[T] and color[p[x]] = Red
4.   do if p[x] = left[p[p[x]]]
5.     then y <- right[p[p[x]]]
6.     if color[y] = Red
7.       then color[p[x]] <- Black
8.         color[y] <- Black
9.         color[p[p[x]]] <- Red
10.        x <- p[p[x]]
11.      else if x = right[p[x]]
12.        then x <- p[x]
13.        Left-Rotate(T,x) ... case 1
14.        color[p[x]] <- Black ... case 2
15.        color[p[p[x]]] <- Red ... case 3
16.        Right-rotate(T,p[p[x]]) ... case 3
17.      else (same as then clause
18.        with "right" and "left" exchanged)
color[root[T]] <- Black
```

تبه کد مربوط به

۴.۶.۳ حذف

برای ساده سازی شرایط مرزی در بیانه سازی RB-Delerr ناگفته‌بین که یک گره محافظتی بنام nil[T] برای درخت T معرفی شائتم. این گره حاوی همان مؤلفه‌هایی می‌باشد که در گردهای معمولی درخت وجود دارد. مؤلفه ریگ این

شی، ساده می‌باشد و سایر مؤلفه‌ها می‌توانند مقادیر دلخواه بگیرند.
با استفاده از این شیء محافظتی می‌توانیم با یک فرزند nil مانند یک گره معمولی درخت که پدرش ۲ می‌باشد
روزگار سایتم (استفاده، مؤلفه Parent مربوط به این شیء محافظتی در فراخوانی RB-Delete-Fixup و اولین اجرای
حلقه While می‌باشد).

نمای این ناید در درخت قرمز- سیاه تمام اشاره گرهای nil را به nil[T] اشاره دهیم (در این مرحله، مؤلفه Parent متناظر با این شیء هنوز مقنن خاص و قابل استفاده‌ای در خود ندارد).

برای حذف یک عضو ایندا برروی مانند حذف از درخت دودویی جستجو، عنصر مؤلفه لظر را حذف و مپس
درخت را تصحیح می‌کیم. ناید وقت کنیم که شیء دستورات مربوط به حذف از درخت قرمز- سیاه با همانی خود در
درخت جستجوی دودویی دقتاً بکسان بست چون در درخت قرمز- سیاه اشاره گرهای nil به nil[T] اشاره می‌کنند و

همجنبین مؤلفه Parent[z] در خط ۷ بدون هیچ شرطی مقدار دهن می‌شود. زیرا در صورتی هم که ریک برگ nil باشد، مقدار nil[T].Parent نتیجه برآیند گرفت. علاوه بر این دو تغییر، در صورتی که ریک لاسیاه باشد، درخت در خطوط ۱۶ و ۱۷ با فراخوانی RB-Delete-Fixup صحیح می‌شود تا خاصیت^۴ در درخت قرمز-سیاه بارگذاری شود.

```

RB-Delete(T,z)
1.   if left[z] = nil[T] or right[z] = nil[T]
2.     then y <- z
3.     else y <- Tree-Successor(z)
4.     if left[y] <> nil[T]
5.       then x <- left[y]
6.       else x <- right[y]
7.       p[x] <- p[y]
8.       if p[y] = nil[T]
9.         then root[T] <- x
10.        else if y = left[p[y]]
11.          then left[p[y]] <- x
12.        else right[p[y]] <- x
13.     if y <> z
14.       then key[z] <- key[y]
15.           if y has other fields, copy them too.
16.     if color[y] = Black
17.       then RB-Delete-Fixup(T,x)
18.   return y

```

شبیه گذ مریوط به RB-DELETE

نکته: اگر RB-Delete با گره z بر روی درخت T فراخوانی شود با خود آن گره حذف می‌شود با عنصر بعدی (Successor) آن حذف می‌شود. در حالت دوم دستور (z) := Key[Successor(z)] بزرگتر خواهد شد.

فرارداد: گره حذف شده را \perp می‌نامیم.

اگر لاسیاه باشد، حذف آن موجب می‌شود مسیری که در گذشته از \perp عبور می‌کرده، یک گره سیاه کمتر از سایر مسیرها داشته باشد و این باعث از بین رفتن خاصیت^۴ در درخت قرمز-سیاه می‌شود. مانند توایم این مشکل را با فرض اینکه توایم گره \perp (که در حقیقت فرزند چپ یا راست گره «قبل از حذف شدن \perp بوده است) را یک بار اضافه می‌نماییم، حل نماییم. یعنی عبور از گره \perp به معنای عبور از دو گره سیاه باشد. بنابراین رمانی که گره \perp را حذف می‌کیم ریک آنرا به فرزندش بعنی \perp می‌فرستیم. تنها مشکل این است که امکان دارد \perp دوبار سیاه شده باشد (خودش از قبل سیاه بوده باشد) و این باعث از بین رفتن خاصیت^۱ در درخت قرمز-سیاه می‌شود.

هدف ما این است که این یک ریک سیاه اضافی را به طرف ارتفاعات بالاتر در درخت بفرستیم تا به یکی از حالت‌های زیر برسیم:

الف) \perp به یک گره قرمز اشاره نماید که در این حالت ما آن گره را سیاه می‌کیم.

ب) \perp به ریشه اشاره کرد که در این حالت سیاه اضافی را می‌توان دور انداخت.

ب) با دورانهای مناسب و رنگ آمیزی مجدد بتوان خاصیت ۱ را احیاء نمود.

اگر در یک مرحله موقع نشویم که به بکی از سه وضعیت بالا برسیم ممکن است با ۴ حالت متفاوت رو برویم
(در حقیقت بر حسب آنکه تر فرزند چپ با راست باشد و وضعیت دوبعدی متقارن بوجود خواهد آمد.)

قراردادها:

- * در هر مرحله ۲ گروهی است که دو برجسته سیاه دارد.

- * برادر (sibling) گره تر خواهد بود.

حالتهایی که ممکن است بوجود باید از قرار زیر است:

۱. قرمز است (شاید فرزندان سیاه دارد.)

۲. سیاه و هر دو فرزندش نیز سیاهند.

۳. سیاه و فرزند چیش قرمز و فرزند راستش سیاه است.

۴. سیاه و فرزند راستش قرمز است.

برای درک بهتر این مطلب، حالتهای مختلف در «شکل ۱۲.۲» آورده شده است.

قراردادها:

- * گروههای سیاه: هر رنگ

- * گروههای قرمز: خاکستری

- * گروههای با رنگ نامعلوم (رنگ این گروههای a و b مشخص شده): سفید

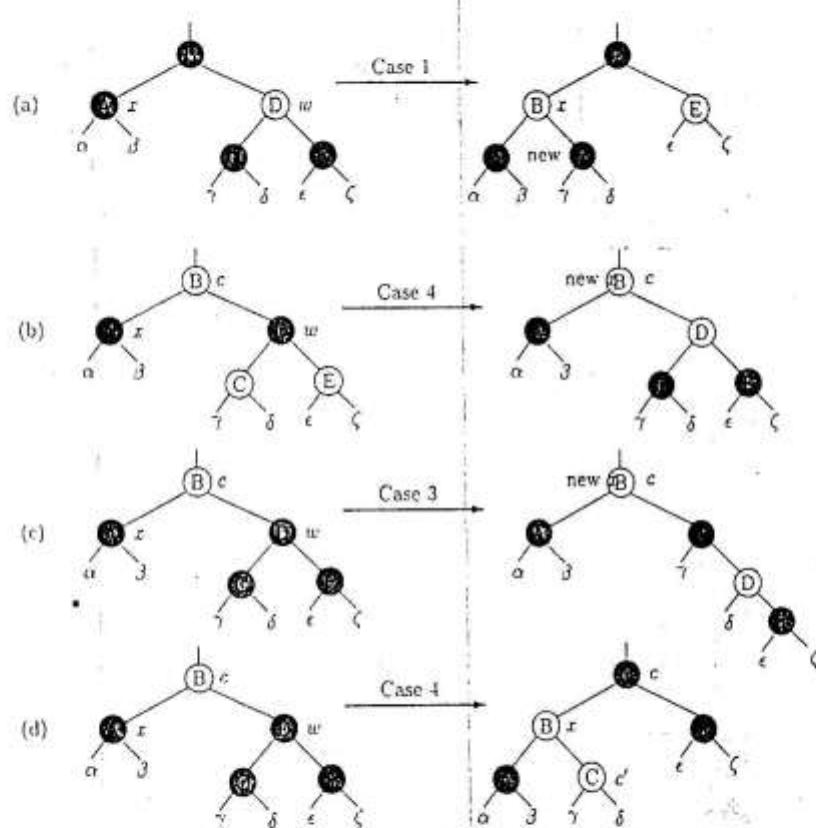
حالت اول: حالت ۱ با تعویض رنگ گروههای D و B با پکدیگر و انجام یک دوران چیگرد به بکی از حالتهای ۲ و ۳ با ۴ تبدل می‌شود.

حالت دوم: رنگ سیاه اضافی که با اشاره گردن نمایش داده شده است با قرمز کردن D و اشاره دادن c به B به یک ارتفاع بالاتر منتقل می‌شود. اگر از طریق حالت اول وارد حالت ۲ شده، باشیم چون c قرمز است حلقه به پایان خواهد رسید.

حالت سوم: حالت ۳ با تعویض رنگ گروههای C و D و انجام یک دوران راستگرد به حالت ۴ تبدل می‌شود.

حالت چهارم: در حالت ۴ رنگ سیاه اضافی که با نمایش داده شده است را می‌توان با عوضی کردن چند رنگ و یک دوران چیگرد بدون آنکه به خواص درخت لطمه‌ای وارد شود حذف نمود و حلقه در این مرحله به پایان می‌رسد.

واضح است که $O(\lg n)$ هم RB-Delete می‌باشد.



شکل ۱۲.۲: حالتهای مختلف ایجاد شده در مراحل حذف

```

3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
RB-Delete-Fixup(T,x)
1. while x <> rot[T] and color[x]=BLACK
2.   do if x = left[p[x]]
3.     then w <- right[p[x]]
4.       if color[w] = RED
5.         then color [w] <- BLACK
6.           color[p[x]] <- RED
7.           Left-Rotate (T, p[x])
8.           w <- right [p[x]]
9.       if color[left[w]]=BLACK and color[right[w]]=BLACK
10.      then color[w] <- RED
11.        x <- p[x]
12.        else if color[right[w]] = BLACK
13.          then color[left[w]] <- BLACK
14.            color[w] <- RED
15.            Right-Rotate(T,w)
16.            w <- right[p[x]]
17.            color[w] <- color[p[x]]
18.            color[p[x]] <- BLACK
19.            color[right[w]] <- BLACK
20.            Left-Rotate(T,p[x])
21.            x <- root[T]
22.        else (same as then with right and left exchanges)
23.        color[x] <- BLACK

```

شبکه کد مریبوط به RB-DELETE-FIXUP

فصل ۴

روش‌های طراحی الگوریتم‌ها

روش‌های طراحی الگوریتم‌ها به منظور حل مسئله‌های مختلف را می‌توان به صورت زیر تقسیم‌بندی کرد:

۱. متنی بر استقرا^۱
۲. تقسیم و حل^۲
۳. برنامه‌ریزی پویا^۳
۴. خریصانه^۴
۵. جستجوی فضای حالت^۵

روش پنجم^۶ جستجوی فضای حالت را به نظم در می‌آورد. برای محدود کردن فضای جستجو، که اغلب بسیار بزرگ است، از «درخت بازی»^۷، به خصوص هرس $\beta - \alpha$ و تیز روش (انشعاع و تحدید)^۸ استفاده می‌شود. در صورتی که راه حل سریع بسیار کند باشد، می‌توان از روش‌های تقریبی یا مکائنه‌ای^۹ استفاده کرد.

۱.۴ استقرا ریاضی

ابنات حکمی برای حالت کلی n به روش استقرای ساده شامل مرحله‌های زیراست:

پایه‌ی استقرا^{۱۰}: ثابت می‌کیم حکم برای $1 = n$ درست است.

فرض استقرا^{۱۱}: فرض می‌کیم حکم برای $1 - n$ درست است.

induction ^۱
divide and conquer ^۲
dynamic programming ^۳
greedy ^۴
state space search ^۵
backtracking ^۶
game tree ^۷
branch and bound ^۸
heuristic ^۹

حکم استقرا^{۱۷}: ثابت می کنیم حکم برای n درست است.

الله استقرا قوی^{۱۷} هم داریم که در آن در فرض استقرا، فرض می کنیم حکم برای $n < i$ درست است، و حکم را برای برای $i = n$ ثابت می کنیم. برای مثال:

پایه: حکم برای $2 \approx n = 1$ درست است.

فرض: حکم برای $2 > n$ درست است.

حکم: برای $2 + 2 = n$ درست است.

چون ثابت کردیم حکم برای $1 = 2 + 2 = n$ درست است پس ثابت می شود که برای تمام اعداد طبیعی هم درست است. و در مثالی دیگر،

پایه: برای $1 = n$ درست است.

فرض: برای $2^k = n$ درست است.

حکم: برای $2^{k+1} = n$ نیز درست است.

بدین ترتیب ثابت می شود که برای کلیه توانهای ۲ درست است.

مثال ۱ - محاسبه فرمول

مقدار $T(n)$ را به صورت فرمولی بر حسب n محاسبه کنید:

$$T(n) = 1 + 12 + 18 + 22 + \dots + (3 + 5n)$$

حدس می زیم که $T(n) = c_1 n^2 + c_2 n + c_3$. برای ثبات، ابتدا مقادیر c_1 , c_2 و c_3 را به دست می آوریم و به کمک استقرا ثابت می کنیم که این دورابطه همانند.

$$\begin{aligned} T(1) &= 1 \Rightarrow c_1 + c_2 + c_3 = 1 \\ T(2) &= 21 \Rightarrow 4c_1 + 2c_2 + c_3 = 21 \\ T(3) &= 39 \Rightarrow 9c_1 + 3c_2 + c_3 = 39 \end{aligned}$$

بنابراین:

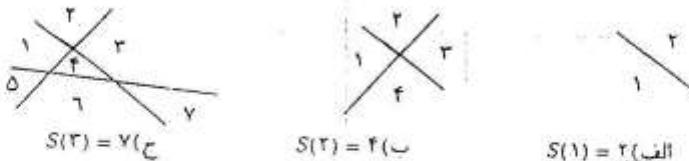
$$\begin{aligned} c_1 &= \frac{3}{2} \\ c_2 &= \frac{11}{2} \\ c_3 &= 0 \end{aligned}$$

یعنی باید ثابت کنیم:

$$T(n) = \frac{3}{2}n^2 + \frac{11}{2}n$$

حال به کمک استقرا حکم بالا را ثابت می کنیم

strong induction^{۱۸}



شکل ۱.۴: شمارش ناجههای بین خطها در یک صفحه

$$T(1) = \lambda$$

$$\text{فرض: } T(n-1) = \frac{2}{7}(n-1)^2 + \frac{11}{7}(n-1)$$

$$\text{حکم: } T(n) = \frac{2}{7}n^2 + \frac{11}{7}n$$

$$\begin{aligned} T(n) &= T(n-1) + 2 + 5n \\ &= \frac{2}{7}(n-1)^2 + \frac{11}{7}(n-1) + 2 + 5n \\ &= \frac{2}{7}(n^2 - 2n + 1) + \frac{11}{7}(n-1) + 2 + 5n \\ &= \frac{2}{7}n^2 - 5n + \frac{2}{7} + \frac{11}{7}n - \frac{11}{7} + 2 + 5n \\ &= \frac{2}{7}n^2 + \frac{11}{7}n \end{aligned}$$

مثال ۲ - شمارش تعداد ناجههای بین n خط در صفحه

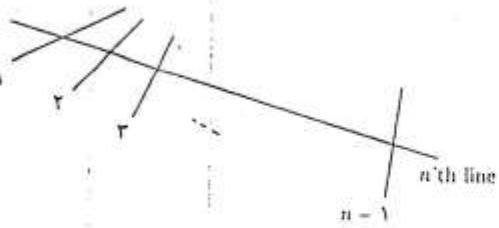
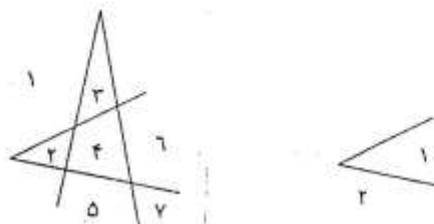
«خط در صفحه‌ای نامتناهی داده شده‌اند. با این فرض که خطها دوپردازی هستند و همچنین هیچ سه خط در یک نقطه پکدیگر را قطع نمی‌کنند، می‌خواهیم تعداد ناجههای بین n خط را محاسبه کیم. برای حل می‌توانیم حدس بزیم که بارسم خط n ام چند ناجه اضافه می‌شود.

$$S(1) = 2$$

$$S(n) - S(n-1) = ?$$

فرض می‌کیم تعداد ناجههای ایجاد شده با $1 - n$ خط $S(n-1) - n$ باشد. وقتی خط n ام را رسم می‌کنیم چون سواری با هیچ یک از خطهای نیست پس هر خط را در یک نقطه قطع می‌کند. روی خط n ام $1 - n$ نقطه‌ی تلاقی با $1 - n$ خط قبلی خواهیم داشت (شکل ۱.۵).

هیچ یک از نقاط تلاقی با خط n ام تکراری نیست چون هیچ سه خطی پکدیگر را در یک نقطه قطع نمی‌کند. بنابراین خط n ام به n پاره خط تقسیم می‌شود. این پاره خطها هر کدام از درون یکی از ناجه‌های فعلی می‌گردند ولذا آن ناجه را به در ناجه‌ی جدید تقسیم می‌کند. پس «ناجهه به ناجه‌هایی که با $1 - n$ خط ایجاد شده‌بود اضافه

شکل ۲.۳: نلاتی خط $n-1$ با پنجه خطها

$$S(1) = 1$$

$$S(n) = 1 + S(n-1)$$

شکل ۲.۴: شمارش ناحیه‌های بین زاویه‌ها در یک صفحه

$$\begin{aligned} S(n) &= S(n-1) + n \\ S(1) &= 1 \end{aligned}$$

$$\begin{aligned} S(n) &= S(n-1) + n - 1 + n \\ &= S(n-2) + n - 2 + n - 1 + n \\ &= S(1) + 1 + 2 + \dots + n \\ &= 1 + 2 + 3 + \dots + n \\ S(n) &= \frac{n(n+1)}{2} + 1 \end{aligned}$$

می‌شود.

مثال ۲ - شمارش حداکثر تعداد ناحیه‌های بین n زاویه

می‌خواهیم حداکثر تعداد نواحی ایجاد شده با رسم n زاویه (با زاویه‌های دلخواه) در یک صفحه را به دست آوریم.

پایه‌ی استقرا: $2 = 1 = D(1)$ بدینهی است.

فرض استقرا: حداکثر تعداد نواحی ایجاد شده با $n-1$ زاویه برابر است با $D(n-1)$. زاویه‌ی n را چنان رسم می‌کنیم که هر ضلع آن هر در ضلع تمامی زاویه‌های قبلی را قطع کند. برای این کار راس این زاویه باید در

۱.۴ استقرای ریاضی

پک ناجهی خارجی انتخاب شود، پعنی ناجهی‌ای که داخل همچ را ویدی قرار نداشته باشد. به عنوان مثال ناجهی ۲ در شکل ۲.۴.الف و ناجهی‌ای ۶ و ۷ در شکل شکل ۲.۴.ب خارجی هستند.

حکم استقرای برای محاسبه‌ی $D(n)$ کافیست مقدار d پعنی تفاضل $D(n)$ و $D(n-1)$ را بپدا کنیم. برای محاسبه‌ی d توجه می‌کنیم که هنگام رسم ضلع اول را ویدی n تعداد نیم خطوط‌ای موجود در صفحه برابر است با $(n-1) \cdot 2$. پس می‌توانیم حداکثر $(n-1) \cdot 2$ نقطه‌ی تلاقی با خط جدید ایجاد کنیم. در این صورت تعداد ناجهی‌های اضافه شده $1 + (n-1) \cdot 2$ خواهد بود. اگرتون تعداد نیم خطوط‌ای موجود در صفحه بر اساس است $1 - 2n + 2n = 4n - 2$. بنابراین با رسم ضلع دوم را ویدی $2n$ ناجهی‌ی دیگر اضافه می‌شود. اما همان طور که در شکل دیده می‌شود به دلیل این که ابتدا دو ضلع وجود ندارد، دو ناجهی از مجموع کل ناجهی‌ها کسر می‌شود. در نتیجه داریم $4n - 2 = d$. حال با باز کردن رابطه‌ی بازگشتی، فرمول دلخواه را به دست می‌آوریم:

$$\begin{aligned} D(n) &= D(n-1) + 4n - 2 \\ &= D(n-2) + 4[n+(n-1)] - 2 \cdot 2 \\ &= D(n-2) + 4[n+(n-1)+(n-2)] - 2 \cdot 2 \\ &\vdots \\ &= D(n-i) + 4[n+(n-1)+(n-2)+\dots+(n-i+1)] - i \cdot 2 \\ &= D(1) + 4[n+(n-1)+(n-2)+\dots+2+1] - (n-1) \cdot 2 \\ &= 2 + 4(n+2)(n-1)/2 - 2(n-1) \\ &= 2n^2 - n + 1 \end{aligned}$$

مثال ۴ - رنگ کردن ناجهی‌های بین خطها

پک صفحه با n خط در آن داده شده است. حداقل با چند رنگ می‌تواند ناجهی‌های ایجاد شده را رنگ کرد به طوری که همچ روناجهی مجاوری پک‌رنگ نشوند؟

پایه: اگر پک با یاره خط داشته باشیم پک طرف آن را رنگ ۱ و طرف دیگر را با رنگ ۲ رنگ می‌کنیم.

فرض: $1 - n$ خط داریم و با دو رنگ ناجهی‌های ایجاد شده بین آنها را رنگ کردیم.

حکم: خط n ام را را رسم می‌کنیم (این خط می‌تواند با پک با چند خط موازی باشد). این خط صفحه را به دو قسمت α و β تقسیم می‌کند. این خط نیز از میان تعدادی ناجهی می‌گذرد و هر کدام را به در ناجه، پکی در قسمت α و دیگری در قسمت β تقسیم می‌کند. کافی است نا رنگ کلیه ناجهی‌های پک قسمت، جه جدید و چه قدمیم، را به رنگ متضاد آنها تغییر دهیم و رنگ‌های ناجهی‌های قسمت دیگر را تغییر ندهیم. بدینهی است که رنگ‌های ناجهی‌های مجاور در هر قسمت پس از این عمل متضاد خواهد بود (با توجه به فرض استقرای) تنها می‌مانند زوج ناجهی‌های مجاوری که با یاره خط مشترکشان قسمتی از خط n ام است. این ناجه‌ها تبلایک ناجه بودند و دارایی پک رنگ؛ با تغییر رنگ همه ناجه‌های پک قسمت، حتماً رنگ‌های این زوج ناجه‌ها هم متضاد خواهند بود.

مثال ۵

هرم زیر موجود است: نه

فصل ۲ روش های طراحی الگوریتم ها

۸۸

$$\begin{array}{ccccccc}
 & & & & & & 1 \\
 & & 3 & + & 5 & & 8 \\
 & & 7 & + & 9 & + & 11 \\
 & & 13 & + & 15 & + & 17 \\
 & & 21 & + & 23 & + & 25 \\
 & & & & & + & 27 \\
 & & & & & + & 29 \\
 & & & & & & 125
 \end{array}$$

مجموع اعداد سطر i ام را بدست آوردید.

$$T(i) = i^r$$

حدس: مجموع اعداد سطر i ام $= i^r$

فرض:

$$\begin{aligned}
 T(1) &= 1 \\
 T(i) &= a_1 + a_r + a_{r+1} + a_{r+2} + \dots + a_i \\
 T(i+1) &= b_1 + b_r + b_{r+1} + b_{r+2} + \dots + b_i + b_{i+1} \\
 b_1 &= a_1 + r \\
 a_r &= a_1 + r \\
 b_1 - a_1 &= ri \\
 b_r - a_r &= ri \\
 &\vdots \\
 b_i - a_i &= ri
 \end{aligned}$$

$$T(i+1) = T(i) + (ri) \times i + b_{i+1}$$

: پس $k = (i+1)(i+r)/2$ کامین عدد فرد است، یعنی b_{i+1} برابر با $1 + 3 + 5 + 7 + \dots + (i+1)$ است.

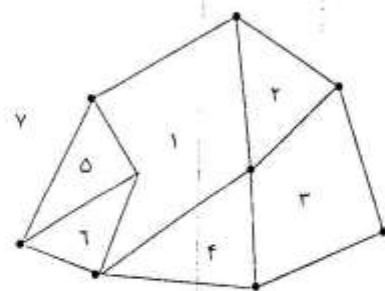
$$b_{i+1} = r \frac{(i+1)(i+r)}{2} - 1 = i^r + ri + 1$$

$$\begin{aligned}
 T(i+1) &= T(i) + ri^r + i^r + ri + 1 \\
 &= i^r + ri^r + ri + 1 \\
 &= (i+1)^r
 \end{aligned}$$

مثال ۱- فرمول اول در گراف های مسطح و همبند

در ابتدا به تعریف های زیر می پردازیم:

۸۸

شکل ۱.۴: پک گراف مسطح با $V = 7$, $E = 9$, و $F = 4$. داریم:

راس منفرد: راسی که هیچ بالی به آن متصل نباشد.

گراف همبند^{۱۴}: گرافی که از هر راس آن به هر راس دیگری لائق یک مسیر وجود داشته باشد.

گراف مسطح^{۱۵}: گرافی که بنوان آن را در صفحه به طوری رسم کرد که بالهای آن هم بگرا نقطع نکنند.

درخت آزاد^{۱۶}: گراف همبند بدون چهت و بدون دور.

مساله - گراف همبند و مسطح با n راس، E اتصال و F ناجه داده شده است ثابت کنید: $V + F = E + 1$.
(به مثال شکل ۱.۴ توجه کنید).

استقرا را روی F انجام می دهیم:
پایه‌ی استقرا: $F = 1$; گراف پک درخت آزاد است.

قضیه ۵. در هر درخت آزاد با n راس و E بال داریم $V = E + 1$.

اینات. اگر درخت را ریشه دار فرض کنیم:

در این صورت به هر راسی، به جزءیش، یک بال وارد شده است، پس تعداد رأس‌ها پکی بیش از تعداد بالهای است. اما برای درخت آزاد در حالت کلی دوباره از استقرا استفاده می‌کنیم (به این کار به اصطلاح استقرا دوبل^{۱۷} گفته می‌شود).

* روش اول: استقرا روی V :

پایه‌ی استقرا: $V = 1$, $E = 1$ بدینهی است.

فرض استقرا: برای $V = n$ رابطه برقرار است.

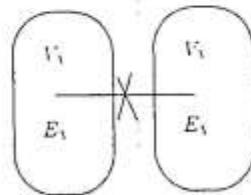
connected graph^{۱۸}
planar graph^{۱۹}
free tree^{۲۰}
double induction^{۲۱}



الف) درخت آزاد

ب) درخت ریشه‌دار

شکل ۴.۴: فرمول اول برای درخت‌ها.



شکل ۶.۴: اثبات فرمول اول برای درخت آزاد.

حکم استقرا: برای E رابطه برقرار است. حتیاً یک راس با درجه ۱ وجود دارد زیرا در غیر این صورت دور ایجاد می‌شود. این راس و بال منصل به آن را حذف می‌کیم. طبق فرض استقرا در گراف حاصل رابطه $V = E - 1 + 1 = E$ برقرار است. با افزودن یک واحد به طرفین معادله داریم: $V = E + 1$.

روش دوم: استقرا روی E

پایه‌ی استقرا: $E = 1, V = 1$ بدهیهی است.

فرض استقرا: برای $E < E'$ رابطه برقرار است. (توجه شود در اینجا از استقرا قوی استفاده می‌کیم.)

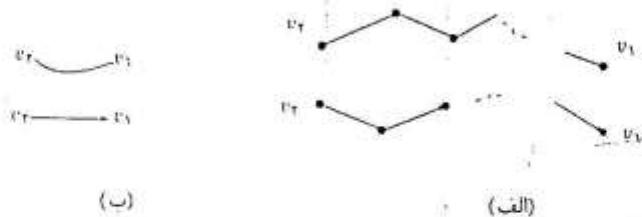
حکم استقرا: برای E رابطه برقرار می‌باشد. یک بال دلخواه را حذف می‌کنیم با توجه به این که در درخت دور وجود ندارد هیچ مسیر دیگری بین دو راس دوسرانین بال وجود نخواهد داشت. بنابراین درخت به دو درخت با تعداد بال‌های کمتر تجزیه می‌شود. فرض کنید تعداد راس‌ها و بال‌های این دو درخت به ترتیب V_1, E_1 و V_2, E_2 باشد.

طبق فرض استقرا داریم: $V_1 = E_1 + 1$ و $V_2 = E_2 + 1$. از جمع این دو رابطه داریم:

$$V = E + 1$$

بنابراین حکم برای $E = F$ ثابت شده است.

فرض استقرای مبالغی اصلی: رابطه برای $V = F$ برقرار است.



شکل ۱.۴-۲: (a) و (b) مجاور هم هستند.

حکم استقراری: رابطه برای F نیز برقرار می باشد:

بک ناجیه وجود دارد که همسایه‌ی ناجیهی خارجی است. این ناجیه در داخل یک دور محاط شده است. حال بک پال که مراز این ناجیه و ناجیه‌ی خارجی باشد را حذف می کنیم. از آن جا که این پال از یک دور حذف شده گراف حاصل هم‌چنان همبند باقی خواهد ماند. با انجام عمل فوق ناجیه‌ی مجاور در ناجیه‌ی خارجی ادغام می شود. پس طبق فرض استقرارا در این گراف جدید داریم: $V + (F - 1) = (E - 1) + 2$. با افزودن بک واحد به طرفین معادله فوق به دست می آوریم: $V + F = E + 2$. و حکم ثابت است.

مثال ۷ - مجموعه‌ی مستقل در گراف‌ها

در ابتدا به تعریف‌های زیر می پردازیم:

دسترسی^{۱۸}: رأس v نوسط رأس u قابل دسترسی است اگر مسیری از v به u وجود داشته باشد.

مجاورت^{۱۹}: رأس v مجاور رأس u است اگر بالی از v به u موجود باشد. توجه کنید که روابط فوق در یک گراف بدون جهت خاصیت تقارنی دارند (شکل ۱.۴-۳).

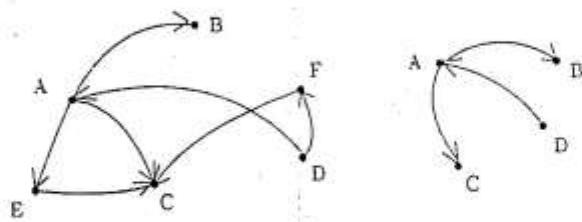
زیرگراف القابی^{۲۰}: در گراف $(V, E) = G$ زیرگراف القابی G_1 با مجموعه‌ی رأس‌های H گرافی است که مجموعه‌ی رأس‌های آن H و مجموعه‌ی بال‌های آن $E \cap H^T$ است، یعنی همه‌ی بال‌هایی از G که دو سرشاران در G_1 است (شکل ۱.۴-۴).

مجموعه‌ی مستقل^{۲۱} در یک گراف: در گراف $(V, E) = G$ زیرمجموعه‌ای از V مثل I بک مجموعه‌ی مستقل است اگر هیچ دور اس موجود در H با هم مجاور نباشد. به عنوان مثال مجموعه‌ی $\{B, E, F\}$ در شکل ۱.۴-۴ بک مجموعه‌ی مستقل است.

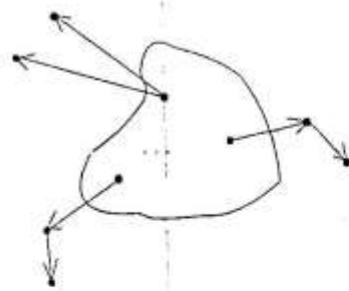
مجموعه‌ی همسایگی^{۲۲}: در گراف $(V, E) = G$ مجموعه‌ی همسایگی بک رأس مثل v به صورت زیر تعریف می شود:

$$N(v) = \{v\} \cup \{w \in V | (v, w) \in E\}$$

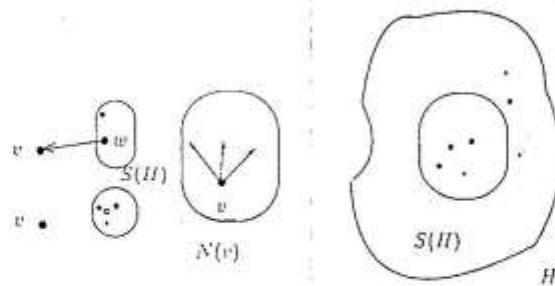
reachability^{۱۸}
adjacency^{۱۹}
induced subgraph^{۲۰}
independent set^{۲۱}
neighbourhood^{۲۲}



شکل ۴.۴: زیرگراف‌هایی G_1 با مجموعه‌ی رأس‌های $\{A, B, C, D\}$ از گراف G .



شکل ۴.۵: مجموعه‌ی مستقل.



شکل ۱.۴.۲: اثبات قضیه.

بعنی مجموعه‌ای شامل خود و تمام رأس‌های مجاورش.

قضیه ۶. برای هر گراف جهت دار $G = (V, E)$ یک مجموعه‌ی مستقل $S(G)$ وجود دارد که در رأس گراف از بکی از رأس‌های این مجموعه‌ی مستقل با سبیری به طول حداقل ۲ قابل دسترسی است.

اثبات. بر روی n (تعداد رأس‌ها) استقراری انجام می‌دهیم.

پایه‌ی استقرار: برای $n = 2$ بدینهی است.

فرض استقرار: برای $n < n'$ قضیه درست است. (از استقراری قوی استفاده می‌کیم).

حکم استقرار: برای n نیز درست است.

راس « v » و زیرگراف القابی H با مجموعه‌ی رأس‌های $(v - N(v))$ را در نظر بگیرید. طبق فرض استقرار H دارای مجموعه‌ی مستقلی مثل $S(H)$ است که خاصیت مطلوب را دارا می‌باشد. حالا دو حالت زیر را بررسی می‌کیم:

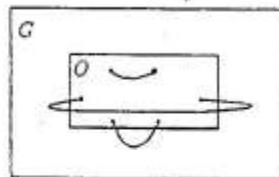
حالت اول – رأس v مجاور بکی از رأس‌های $S(H)$ مثل w است:

در این صورت همان مجموعه‌ی $S(H)$ جواب است $S(H) = S(G)$. زیرا همه‌ی رأس‌های موجود در $N(v)$ از w با سبیری به طول حداقل ۲ قابل دسترسی هستند.

حالت دوم – رأس v مجاور هیچ کدام از رأس‌های $S(H)$ نیست:

در این صورت با توجه به این که هیچ یک از رأس‌های $S(H)$ نیز مجاور v نیست می‌توان v را به $S(H)$ اضافه کرد و مجموعه‌ی حاصل هم‌چنان مستقل خواهد بود. و مابقی رأس‌های موجود در $(v - N(v))$ اکنون از v با سبیری به طول ۱ قابل دسترسی خواهند بود. بنابراین $S(H) = S(G)$ مجموعه‌ی خواسته شده است.

برای مثال در شکل ۱.۴.۳ می‌توانیم $S(G)$ را برای مجموعه‌ی D بگریم.



شکل ۱۱.۴: مسیرهای مستقل بالی.

یک مسئله با استفاده از قضیه‌ی فوق

الگوریتمی بوسیله که گره‌های را در یک گراف داده شده رنگ کند به طوری که هیچ یک مجاور دیگری نبوده و فاصله‌ی هر گره گراف از یکی از آن‌ها کمتر از ۳ باشد.

روش حل: یک گره‌ی دلخواه را در نظر گرفته و آن گره و کلیه‌ی رأس‌های مجاورش را از گراف حذف می‌کیم. به صورت بازگشتی مسئله را برای گراف کوچکتر حاصل حل می‌کیم. ختم بازگشت و قدمی است که تعداد رأس‌ها کمتر از ۲ شود. حال اگر گره‌ی حذف شده مجاور هیچ یک از رأس‌های گراف جدید نیاشد آن را نیز رنگ می‌کیم. گراف را می‌توان به عنوان یک شهر و رأس‌های رنگ شده را به عنوان مراکز آتش نشانی در نظر گرفت.

مثال ۸ - مسیرهای مستقل بالی

تعریف مسیرهای مستقل بالی^{۷۲}: دو مسیر در یک گراف مستقل بالی هستند اگر دارای بال مشترک نباشند.

قضیه ۷. در گراف $(V, E) = G$ فرض کنید O مجموعه‌ی رأس‌های با درجه‌ی فرد باشد. O را می‌توان به زوج رأس‌های بتدبیل کرد که مسیرهای بین این زوج رأس‌ها مستقل بالی باشند.

شکل ۱۱.۴ مثالی از این قضیه است.

اثبات: در مجموعه‌ی فوق (O) رابطه‌ی زیربرقرار است:

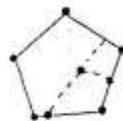
$$|O| = 2k$$

ابتدا رابطه‌ی فوق را اثبات می‌کیم. می‌دانیم که $\sum_{v \in V} \deg(v) = 2k$ یعنی عدد زوج است چون برای هر یال موجود در گراف، دو درجه به این مجموع اضافه می‌کند. از طرف دیگر هر راس موجود در مجموعه‌ی O از درجه‌ی زوج است بنابراین $\sum_{v \in O} \deg(v) = 2k$ عدد زوجی است. چون:

$$\sum_{v \in V} \deg(v) = \sum_{w \in V - O} \deg(w) + \sum_{u \in O} \deg(u)$$

چون برای هر $v \in V$ عدد فردی است $\deg(v) = u_i \in O$ ، u_i ها باید زوج باشد یعنی رابطه‌ی فوق الذکر صحیح است.

^{۷۲}Edge Disjoint



شکل ۱۲.۴: اثبات غلط قضیه اول.

حال با استقرا روی تعداد پال‌ها قضیه را ثابت می‌کیم؛ ایندا فرض می‌کنیم که گراف همیند نیست. بنابراین گراف مورد نظر ما به چند گراف همیند کوچکتر تقسیم می‌شود که برای هر گراف همیند قضیه صادق است (با به فرض استقرا). بنابراین برای کل گراف نیز صحیح است.

حال گراف G را در نظر بگیرید که $V = 5$ و $E = 5$ دو عضو O باشد و همیند باشد مسیری از v_1 به v_5 وجود دارد که با حذف این مسیر ممکن است G به یک گراف غیر همیند تبدیل شود که در آن گراف مجموعه‌ی رئوس با درجه‌ی فرد برابر است با: $\{v_1, v_2\} - O$ که با فرض استقرا قضیه برای این گراف صادق است. در غیر این صورت با حذف این مسیر گراف همیند می‌ماند یعنی برای هر دوران مسیری بین آن دو هست که بار می‌توان عمل فوق را نکرار کرد.

□

۱.۱.۴ خطاهای معمول در اثبات با استقرا

به چند نوع خطاهایی که ممکن است در استفاده از استقرا ایجاد شود توجه کنید:

۱. در اثبات فرمول اول ($V + F = E + 2$) شخصی این قضیه را این چنین اثبات می‌کند:

اثبات به طریقه استقرا: یک وجه داریم، از وسط یک یال به وسط پال دیگری یک پال وصل می‌کنیم
(شکل ۱۲.۴).

V دو واحد اضافه شده؛

F یک واحد اضافه شده؛ و

E سه واحد اضافه شده است.

پس رابطه‌ی مذبور صحیح است.

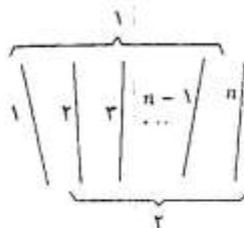
اشکال این اثبات در این جاست که اگر از پایه‌ی استقرا شروع کنیم و با استفاده از حکم مرحله به مرحله گراف را بزرگتر کنیم، ممکن است گراف داده شده تولید نشود، چرا که رأسی که اضافه می‌شود همیشه از درجه‌ی فرد است.

۲. خط در صفحه داریم ثابت می‌کیم که همگی از یک نقطه می‌گذرند.

اثبات به طریقه استقرا:

پایه: برای $n = 2$ مسئله صحیح است.

فرض: برای $1 - n$ خط مسئله صحیح است.



شکل ۱۲.۴: اثبات غلط با استفرا.

جکم: برای «خط مسئله صحیح است.

مطابق شکل ۱۲.۴ - ۱ - «خط شماره‌ی ۱» با به فرض از یک نقطه می‌گذرد. ۱ - «خط شماره‌ی ۱۱

هم بنا به فرض از یک نقطه می‌گذرد. این دو نقطه باید یکی باشند چون این دو دسته خط شامل خط‌های مشترک‌کند. بنابراین کل «خط از یک نقطه می‌گذرد.

اشکال اثبات در پایه‌ی استفرا است، پایه باید $2 = n$ باشد، نه $n = 2$ و برای $2 = n$ ، پایه درست نیست!

۳. ثابت کنید که

$$n = \sqrt{1 + (n-1)\sqrt{1 + (n)\sqrt{1 + \dots}}}$$

اثبات به وسیله‌ی استفرا :

پایه : برای $1 = n$ مسئله صحیح است.

ظرفین فرمول فوق را به توان دو می‌رسانیم، داریم:

$$n^2 - 1 = (n-1)\sqrt{1 + (n)\sqrt{\dots}} \Rightarrow n+1 = \sqrt{1 + (n)\sqrt{\dots}}$$

که همان فرمول فوق برای $1 + n$ است و مسئله اثبات شده است.

اشکال اثبات در این جاست که ما بر $1 - n$ تقسیم کردیم در حالی که پایه‌ی استفرا $1 = n$ است؛ یعنی که در همان مرحله‌ی اول ما بر صفر تقسیم کردیم که اشکال دارد.

٢.٤ طراحی الگوریتم با استقرا

برای حل مسائلی که ماهیت بازگشتی دارند، استفرا بک روش سیار قوی است. به کمک استفرا می‌توان برای چنین مسائلی الگوریتم مناسب طراحی کرد، درستی الگوریتم را ثابت و آن را تحلیل کرد. در اینجا برای چند مسئله‌ی این مرحله‌ها را نشان می‌دهیم.

۱۲۰۴

کسی خواهیم به n شنی کدھای متلبی اختصاص دهیم. ساده‌ترین روش برای این کار آن است که به آنها اعداد ۱ تا n اختصاص دهیم و با توجه به آن که نام کارهای ما در سیستم دودویی انعام می‌گیرد این n عدد را به صورت دودویی نمایش دهیم. بدینهی است که در این حالت تعداد بیت‌ها برای هر کد برابر است با $\lceil \log_2 n \rceil$ که به قدر

ولی به علت محدودیت‌های ساخت‌افزاری در شمارنده‌ها می‌خواهیم که کدهای متوازن فقط در یک بیت اختلاف داشته باشد. یکی از این دلایل این محدودیت جلوگیری از ایجاد Hazard در مدارهای منطقی‌ای است که این کدها ایشت سر هم می‌شوند.

بنابراین مسئله که می‌خواهیم حل کنیم این است: چگونه می‌توان به n شیء که دو دوی با تعداد بیت کمینه (یعنی $\log_2 n$ بیت) اختصاص داد تا کدهای متواالی فقط در یک بیت اختلاف داشته باشند؟ توجه کنید که این توالی کدرا کدها «دورانی» است؛ یعنی هر کد دقیقاً در کد قبل و بعد دارد. به چنین کدی «کد گردی^{۲۰}» می‌گویند.

کد گری بسته ۲۵: هرگاه هر کد با هر کدام از دو کد بعدی و قبلی اش دقیقاً در یک بیت اختلاف داشته باشد. بنابراین کدها تشکیل یک دور می‌دهند. مثلًا، $1110 \rightarrow 1110 \rightarrow 0111 \rightarrow 0011 \rightarrow 0001$ و $1000 \rightarrow 1011 \rightarrow 1111 \rightarrow 1011 \rightarrow 1010$. یک کد بسته برای ۱۰ عنصر است. توجه کنید که ۰۰۰۰ و ۱۰۰۰ هم دو یک است با هم اختلاف ندارند.

کد گری باز^{۲۱}: همان کد بسته است مگرین تنها دو گد متولی که فقط اختلاف آن دو بیش از پک بیت است.
مثال: $1010 \rightarrow 1011 \rightarrow 1110 \rightarrow 1111 \rightarrow 1110 \rightarrow 0111 \rightarrow 0010 \rightarrow 0000$

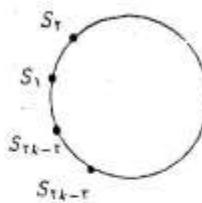
- ¹⁰ See also the discussion in section 3.

- $\{(\alpha \rightarrow \beta) \rightarrow (\gamma \rightarrow \beta)\} : \vdash \alpha \rightarrow \beta$ (using $\alpha \rightarrow \beta$ as a premise)

- $(\ast \ast \Rightarrow \ast) \Rightarrow \lambda x. (\sim \lambda z. s_1 z, \lambda b. s_2 \sim b \rightarrow \lambda y. t \rightarrow s_3 y) \equiv T \cdot s_1 \cdot s_2 \cdot T$

^۴. ثابت کرد که برای تعداد زوج، عناصر ($n = 2k$) می‌توان کدگیری بسته ایجاد کرد.

Gray code¹¹
close Gray code¹²
Open Gray Code¹³



شکل ۱۴.۲: تعابش کد گری

اثبات. این لم را به کمک استغرا به صورت «سازنده»^{۱۷} اثبات می‌کیم؛ یعنی هم اثبات می‌کیم که این کارشناسی است و هم نشان می‌دهیم که چگونه و با چه الگوریتمی می‌توان کدها را تولید کرد.

پایه‌ی استغرا: $2 = 2 \cdot 1 = n$ که بدینهی است.

فرض استغرا: برای $2k - 2 = 2k = n$ کد بسته وجود دارد.

حکم استغرا: برای $n = 2k$ هم کد بسته داریم.

با به فرض استغرا برای $2 - 2k = n$ کد بسته‌ی $(S_1, S_2, \dots, S_{2k-2})$ را داریم که می‌توان آن را به صورت دور ستهای مانند شکل زیر نشان داد. رابطه‌ی بین دو کد متعابز را که در پک بیت با هم اختلاف دارند را با خط تیره نشان می‌دهیم (شکل ۱۴.۴).

ما پک بیت صفر به همه‌ی کدهای موجود اضافه کیم؛ باز هم رابطه‌ی فوق برقرار است. کدهای S_{2k-1} و S_{2k} را به این صورت اضافه می‌کیم؛ کد S_{2k} همان کد S_1 است، با این اختلاف که بیت صفر اضافه شده به S_1 یک پک باشد. کد S_{2k-1} همان کد S_{2k-2} باشد با این اختلاف که بیت صفر اضافه شده به S_{2k-2} یک پک باشد. با این ترتیب، با S_{2k-2} و S_{2k-1} می‌باشد S_{2k-1} با S_{2k} فقط در پک بیت اختلاف دارند. بنابراین کد حاصل برای $n = 2k$ هم بسته است. \square

توجه: در اثبات فوق الگوریتمی از ایده دادیم که در آن، برای اضافه کردن هر دو عنصر به مجموعه عناصر پک بیت به کدهای قبلی اضافه شد. بنابراین با این الگوریتم، برای $n = 2k$ عنصر کد بسته‌ای با $\frac{n}{2}$ بیت ایجاد می‌کیم که بدینهی نیست.

لم ۵. برای $2^k = n$ عنصر می‌توان کد گری بسته‌ی k بیتی ایجاد نمود.

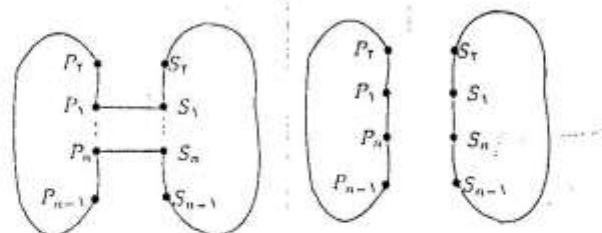
اثبات. اثبات به کمک استغرا.

پایه: برای $2 = n$ بدینهی است.

فرض: برای $2^{k-1} = n$ کد بسته‌ی $k-1$ بیتی وجود دارد.

حکم: برای $2^k = n$ کد بسته‌ی k بیتی وجود دارد.

constructive^{۱۸}



شکل ۲.۴: ساخت کد گری به طول دو برابر

مجموعه‌ای کدهای بسته‌ی $k - 1$ بیتی فرض استفرا (برای $2^{k-1} = n$ عنصر) را $\{S_1, \dots, S_n\} = S$ می‌نامیم. فرض کنید $P = \{P_1, \dots, P_n\}$ یک کد دیگر مشابه‌ی S است (کدهای P_i و S_i بکان هستند). این دو کد بسته را به صورت دور می‌توان نشان داد (شکل ۲.۴).

به دور S یک بیت ۱ و به دور P یک بیت ۰ (صفر) اضافه می‌کنیم تا کدهای $\{S'_1, \dots, S'_{n+1}\} = S'$ و $\{P'_1, \dots, P'_{n+1}\} = P'$ ایجاد شوند. بدینهی است که مجموعه‌ی $\{P'_1, \dots, P'_{n+1}, S'_1, \dots, S'_{n+1}\}$ یک کد بسته و بهینه‌ی k بیتی برای $n = 2^k$ عنصر است.

لم ۶. برای $1 \leq k \leq n = 2^k$ عنصر، کد گری بسته وجود ندارد.

اثبات. فرض کنید که برای این تعداد عناصر یک کد گری بسته داشته باشیم. در این صورت، اگر از یک کد شروع کنیم در پک دور زدن باید به همان کد برسیم. با توجه به آن که از یک کد به کد بعدی فقط یک بیت تغییر می‌کند، باید تعداد تغییرات این بیت‌ها زوج باشد تا نهان به همان کد اولیه برسیم، یعنی اگر یک بیت صفر به پک تبدیل شود باید در جای دیگر همان بیت یک دوباره به صفر تبدیل شود. ولی این کار با توجه به تعداد فرد عناصر ممکن نیست.

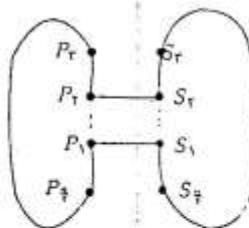
□

قضیه ۸. برای $1 \leq k \leq n$ کد گری $\log_2 k$ بیتی ایجاد کرد. اگر «زوج باشد این کد بسته و اگر» «فرد باشد باز است.

اثبات. با استفرا اثبات می‌کنیم.

حالات اول: فرض می‌کنیم تعداد عناصر زوج است ($n = 2k$).

$k = \frac{n}{2}$ زوج است با فرد. طبق فرض استفرا، اگر k زوج باشد: برای k عنصر کد گری بسته، و اگر فرد باشد، کد گری باز، هر دو به طول $\log_2 k$ بیت وجود دارد. اگر k زوج باشد، درست مانند اثبات لم ۵ می‌توان برای n عنصر کد گری بسته‌ی $\log_2 k + 1 = \lceil \log_2 n \rceil$ بیتی ایجاد کرد. اگر k فرد باشد، دو کد باز و مشابه‌ی $S = \{S_1, \dots, S_k\}$ و $P = \{P_1, \dots, P_k\}$ را برای k عنصر در نظر بگیرید، و فرض کنید که تنها کدهای S و S_r (و نیز P_r و P_r) با هم در بین از یک بیت اختلاف دارند. این دو کد را می‌توان به صورت دو دور «ماهار» مانند شکل ۲.۴ نشان داد.



شکل ۱۶.۴: ساخت گدگری

به دور S پک بیت ۱ و به دور P پک بیت ۰ (صفرا) اضافه می‌کنیم تا کدهای $\{S'_1, \dots, S'_n\}$ و $\{P'_1, \dots, P'_n\}$ باشد شوند. بدینهی است که مجموعه‌ی $\{S'_1, \dots, S'_n, P'_1, \dots, P'_{n-1}, P'_n\}$ پک گد بسته است و تعداد بیت‌های آن برابر است با

$$\lceil \log_2 \frac{n}{\gamma} \rceil + 1 = \lceil \log_2 n \rceil$$

حالت دوم: فرض می‌کنیم تعداد عناصر فرد است ($n = 2k + 1$)

برای $(2k + 2)$ عنصر پک گد بسته‌ی $\lceil \log_2 2k + 2 \rceil$ بینی ایجاد می‌کنیم. کافی است پک عنصر دلخواه را حذف کنیم؛ گد بسته به گد باز تبدیل و با طول بهینه تبدیل می‌شود، چون $\lceil \log_2 2k + 2 \rceil = \lceil \log_2 2k + 1 \rceil$

۲.۲.۴ رابطه‌ی مستقل از حلقه برای اثبات درستی الگوریتم

استنزا، یک راه خوب برای اثبات الگوریتم‌هاست. برای مثال اگر یک الگوریتم حاوی یک حلقه باشد، می‌توانیم بالاسفرا روی متغیر حلقه، درستی تتجه را ثابت کنیم. در این صورت رابطه‌ی استقرایی برایهی متغیرهای حلقه به گونه‌ی مستقلی از حلقه (Loop Invariant) بیان می‌گردد. این رابطه را اگر برای همه‌ی مقادیر حلقه اثبات کنیم، در واقع درستی آن را اثبات کردہ‌ایم.

جهت روشن تر شدن مطلب به مثال زیر توجه کنید:

```

Procedure Convert_to_Binary(n);
    Input : n (A positive number);
    Output : b (Array of bits);
Begin
    t := n;
    k := 0;
    While t > 0 do Begin
        k := k+1;
        b(k) := t mod 2;
        t := t div 2;
    End;
End;

```

الگوریتم فوق عمل تبدیل عدد n را به رشته‌ی دودویی معادل را بر عهده دارد.
برای اثبات درستی این الگوریتم، برروی k ، تعداد دفعاتی که حلقه تکرار شده است، استقرار انجام می‌دهیم. اما قبل از آن باید رابطه‌ای مستقل از حلقه بیاییم که وابسته به متغیرهای حلقه (از جمله k) باشد و استقرار را برای آن برای k های متفاوت انجام دهیم. به رابطه‌ی ذکر شده invariant می‌گوییم:
در این الگوریتم رابطه‌ی استقراری را به صورت زیر حدس می‌زنیم:

$$n = t2^k + m$$

که در آن m عددی است که آرایه‌ی b نمایش می‌دهد.
اثبات.

$$n = t2^k + m \quad \text{پس } m = n - t2^k$$

$$n = t2^k + m \quad \text{آن‌گاه } k = i \quad \text{فرض استقرار: اگر } k = i \quad \text{آن‌گاه } n = t2^i + m$$

حکم استقرار: اگر $i+1$ باشد $k = i+1$ آن‌گاه $n = t'2^{i+1} + m'$ که t' و m' با رابطه‌ی زیر بیان می‌گردند:

$$t' = \lfloor \frac{t}{2} \rfloor \quad m' = m + (t \bmod 2)$$

با توجه به روابط فوق و فرض استقرار داریم:

$$n = \lfloor \frac{t}{2} \rfloor 2^{i+1} + m + (t \bmod 2)2^i = 2^i(\lfloor \frac{t}{2} \rfloor + (t \bmod 2)) + m$$

از طرفی

$$2^i(\lfloor \frac{t}{2} \rfloor + (t \bmod 2)) = t$$

پس داریم:

$$n = t2^i + m$$

و این رابطه بنا به فرض استقرار صحیح است.

۴.۲.۲.۱ محاسبه‌ی مقدار یک چندجمله‌ای

هدف محاسبه‌ی مقدار یک چندجمله‌ای $P_n(x)$ از درجه‌ی n بر حسب x است.

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

ورودی: a_0, a_1, \dots, a_n, x

خروجی: مقدار $P_n(x)$

راه حل اول: چندجمله‌ای را به صورت بازگشتنی تعریف می‌کنیم: $P_n(x) = P_{n-1}(x) + a_n x^n$ و مطابق آن الگوریتم بازگشتنی برای حل مسئله می‌نویسیم. در این صورت، هرینه‌ی الگوریتم برابر $T(n) = n + 1$ عمل جمع خواهد بود به اضافه‌ی تعداد عملیات‌های ضرب که برابر است با:

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

راه حل دوم: (الگوریتم هورنر^{۷۸})

تعریف را به این صورت تغییر می‌دهیم:

$$P'_{n-1}(x) = a_n x^{n-1} + a_{n-1} x^{n-2} + \cdots + a_1 x + a_0$$

$$P_n(x) = x P'_{n-1}(x) + a_0$$

بنابراین:

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

```

begin
    p=0;
    for i:=n downto 0 do
        p:=p*x+a[i];
end;

```

۴.۲.۲.۲ زیرگراف القایی پیشنهادی

هدف به دست آوردن بزرگترین زیرگرافی است که درجه‌ی هر رأس آن حداقل k باشد. به این زیرگراف «زیرگراف القایی پیشنهادی»^{۷۹} می‌گوییم.

Hornor's algorithm⁷⁸
Maximal Induced Subgraph⁷⁹

۲.۴ طراحی الگوریتم با استغرا

یک مثال کاربردی برای این مسئله: عده‌ای را می‌خواهیم به یک مهمانی دعوت کنیم. هر یک از این افراد تعدادی از دیگران را می‌شناسد و رابطه‌ی «شناسایی» رابطه‌ای دوطرفه است. می‌خواهیم ازین افراد، می‌خواهیم بیشترین تعدادی را دعوت کنیم که هر کدام‌شان لاقل k نفر دیگر را بشناسد. مسئله به صورت گراف مدل می‌شود و هدف به دست آوردن زیرگراف القابی است با درجه‌ی حداقل k .

به عبارت دیگر، در گراف $(V, E) = G$ زیرمجموعه‌ی $S \subseteq V$ را با حداقل تعداد رأس‌های می‌خواهیم بیندازیم که زیرگراف القابی آن دارای رأس‌هایی با درجه‌ی حداقل k باشد.

روش پیاده‌سازی: استفاده از ماتریس مجاورت

حل: اگر درجه‌ی همه‌ی گره‌ها از k بیشتر باشد مسئله حل شده است، در غیر این صورت گره‌ای مانند « وجود ذاره به طوری که $\deg(v) < k$ » Degree(v) مورد نظر و بال‌های مربوط به آن را حذف می‌کنیم. این کار را ادامه می‌دهیم تا وقتی که درجه‌ی هر گره‌ی باقی‌مانده حداقل k باشد.

۵.۲.۴ نگاشت یک به یک

مجموعه‌ی A و نابغ $A \rightarrow A$: f داده شده است. می‌خواهیم بزرگترین زیرمجموعه‌ی A , $C \subseteq A$ را بیندازیم که f بر روی آن زیرمجموعه یک به یک باشد، یعنی $f: C \rightarrow S \subseteq A$

نابغ f بر روی A را می‌توان به صورت یک گراف سودار نشان داد. شکل زیر یک نابغ را بر روی $\{1, 2, 3, 4, 5, 6, 7\} = A$ نشان می‌دهد.

هدف پیدا کردن بزرگترین زیرگرافی است که درجه‌ی ورودی و خروجی هر گره در آن ۱ باشد. این مسئله را می‌توان با استغرا حل کرد.

اگر درجه‌ی ورودی گره‌های این گراف را به دست آوریم، باید گره‌ای به نام $A \in V$ با درجه‌ی ورودی صفر وجود داشته باشد، در غیر این صورت $S = A$ جواب است (چرا؟) نمی‌تواند عضوی از جواب باشد، پس V و بال‌هایی که از آن خارج می‌شوند را حذف می‌کنیم و در گراف حاصل مسئله را به صورت استغرا حل می‌کنیم.

برنامه‌ی زیر این کار را انجام می‌دهد.

```

Input : f{array[1..n] where values are 1..n}
Output: S a subset of A={1..n}

begin
  S:=A;
  for j:=1 to n C[j]:=0; {C[j] is the indegree of j}
  for j:=1 to n Inc(C[F[j]]));
  for j:=1 to n if C[j]=0 then put j in a queue
  while queue is not empty do
    begin
      remove i from queue;
      S:=S-{i};
      Dec(C[F[i]]);
      if C[F[i]]=0 then put F[i] in queue;
    end;
end;

```

۶.۲.۴ مسئله‌ی «ستاره‌ی مشهور»

می‌خواهیم با حداقل تعداد پرسن از نفر n ، فردی که ویزگی‌های پک «ستاره‌ی مشهور»^{۲۰} را دارد، در صورت وجود، پیدا کیم. یک نفر سtarه است اگر بقیه اورا از قبیل بشناسند و او با آن‌ها آشنا باشد. ما مجاز هستیم از n بپرسیم که آیا نارامی شناسد؟ این یک پرسش است. توجه کنید در این مسئله رابطه‌ی شاخن پک رابطه است. چون $n/2 - (n/2 - 1)$ گروه دو نفری داریم پس اگر پرسش‌ها به طور دلخواه باشند، در بدترین حالت نیاز به $(1 - (n/2 - 1))$ پرسش داریم.

راه حل اول: فرض کنیم که ما می‌توانیم ستاره را بین $1 - n$ نفر اول توسط استقرار به دست آوریم. چون حداقل پک ستاره می‌توانیم داشته باشیم، سه حالت ممکن است اتفاق بیفتد:

۱. ستاره بین $1 - n$ نفر اول است.

۲. نفر n ام ستاره است.

۳. ستاره نداریم.

در حالت اول فقط باید بررسی کنیم که نفر n ام ستاره را می‌شناسد و ستاره اورا نمی‌شناسد. در دو حالت بعد، حداقل $n - (1 - (n/2 - 1))$ پرسش نیاز داریم؛ چرا که باید روش کنیم که آیا هر یک از $1 - n$ نفر بقیه نفر n ام می‌شناسد و نفر n ام اورا نمی‌شناسد. بنابراین، جمع کل پرسش‌ها $(1 - (n/2 - 1))$ است که در بدترین حالت هم به آن رسیدیم.

راه حل بهتر: به روش حدیقی عمل می‌کنیم، در هر پرسش یک نفر را از مجموعه حذف کنیم و با استفاده از استقراره حل را دنبال می‌کنیم، برای این کار فرض کنید که ما از A بپرسیم که آیا B را می‌شناسد یا نه؟ اگر جواب مثبت باشد، A نمی‌تواند ستاره باشد، و اگر جواب منفی باشد B نمی‌تواند ستاره باشد؛ در هر صورت یکی از این افراد

celebrity^{۲۰}

۲.۴ طراحی الگوریتم با استغرا

حذف می شوند. حال کافی است بین $1 \sim n$ نظریاتی مانده ستاره را پیدا کنیم. با $1 \sim n$ پرسش به یک نفریه نام * می روییم؛ تنها * می تواند ستاره باشد، و با $\{1 \sim n\}$ پرسش این امر را می توان مشخص کرد. بنابراین تعداد کل پرسش ها می شود:

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

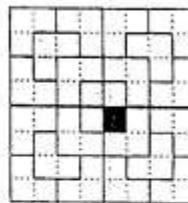
البته این تعداد پرسش ها کمی نیست و می توان کمی بعتر از این عمل کرد.
حال الگوریتم این راه حل را می روییم. ورودی Know یک ماتریس مجاورت $n \times n$ است. مقدار $Know[i, j]$ برابر یک است اگر قدر i فرد j را بشناسد، و گرنه صفر است. هدف پیدا کردن شمارهی s است به گونه ای که تمامی عناصر سطون n ام (به جز $[s, s]$) یک و تمامی عناصر سطر n ام (به جزو لفه (s, s)) صفر باشند:

```

Algorithm celebrity (Know);
Input:Know (an n*n Boolean matrix).
Output:celebrity.

begin
    i:=1;
    j:=2;
    next:=3;
    {in the first phase we eliminate all but one candidate}
    while next <= n+1 do
        if Know[i,j] then i:=next
        else j:=next;
        next:=next+1;
        {one of either i or j is eliminated}
    if i=n+1 then candidate:=j;
    else candidate:=i;
    {Now we check that the candidate is indeed the celebrity}
    wrong:=false;
    k:=1;
    Know[candidate,candidate]:=false;
    {a dummy variable to pass the test}
    while not wrong and k<=n do
        if Know[candidate,k] then wrong:=true;
        if not Know[k,candidate] then
            if candidate<>k then wrong:=true;
            k:=k+1;
        if not wrong then celebrity:=candidate
        else celebrity:=0 {no celebrity}
    end;

```



شکل ۴.۲۷.۲: فرش کردن صفحه‌ی شطرنجی.

۴.۳ روش تقسیم و حل برای طراحی الگوریتم‌ها

این روش که معمولاً برای بدست آوردن الگوریتم‌های سریع در حل مسئله‌ها مورد استفاده قرار می‌گیرد، دارای دو مرحله‌ی « تقسیم » و « حل » است. در مرحله‌ی اول، مسئله به چند زیرمسئله‌ی کوچک‌تر تقسیم می‌شود. این زیرمسئله‌ها به صورت بازگشتی حل می‌شوند و از ترکیب حل آن‌ها جواب مسئله به دست می‌آید. واژه‌ی انگلیسی این روش divide and conquer است که ترجیحه‌ی « تقغیره بینداز و حکومت کن » برای آن بسیار آشناست.

این روش مشابه‌ی روش طراحی الگوریتم با استفاده از استقراس است. البته معمولاً در روش استقرایی مسئله به پک مسئله با اندازه‌ی کوچک‌تر تقسیم می‌شود.

توطیحات بیشتر را در قالب مثال‌های ارلیه می‌کنیم.

۴.۳.۱ فرش کردن صفحه‌ی شطرنجو

صفحه‌ی شطرنجی به ابعاد $2^k \times 2^k$ و موزاییک به شکل ۴.۲۸ داریم. می‌خواهیم این صفحه را طوری با این موزاییک فرش کنیم. چون برای هر k داریم $1 = 2^{k-1} + 2^k + 2^{k-1}$ ، این صفحه را اگر فرش کنیم در بهترین حالت یکی از خانه‌ها خالی می‌ماند. می‌خواهیم نشان دهیم که می‌توان بالانتخاب هر خانه‌ی دلخواه به عنوان خانه‌ی خالی، یقینی صفحه را می‌توان کاملاً فرش کرد، به گونه‌ای که هیچ موزاییکی از جدول بیرون نزند. البته در این فرش کردن ما محاز به دوران موزاییک‌ها هستیم.

راه حل. بر روی k استقرای انجام می‌دهیم. برای $1 = 2^{k-1} + 2^k + 2^{k-1}$ بدهیهی است با چرخاندن موزاییک به جواب می‌رسیم. مربع $2^k \times 2^k$ را مطابق شکل ۴.۲۹ به چهار قسم تقسیم می‌کنیم. طبق استقرای هر مربع را می‌توانیم طوری فرش کنیم که هر خانه‌ی دلخواه آن خالی بماند. خانه‌ی خالی خواسته شده در مسئله درون یکی از این چهار مربع است. این مربع به اندازه $2^{k-1} \times 2^{k-1}$ را با استقرای طوری فرش می‌کنیم تا خانه‌ی مورد نظر خالی بماند. حال سه مربع دیگر را طوری فرش می‌کنیم تا خانه‌ی گوشی محل تلاقی آن‌ها خالی بماند، سه خانه‌ی خالی باقی مانده را بین نویسط یک موزاییک فرش می‌کنیم و مساله حل است.

۴.۳.۲ زمان‌بندی دوره‌ی بازی‌ها

برای n گیم می‌خواهیم یک دوره‌ی بازی با حداقل مدت طراحی کنیم که در آن هر تیم با یقینی تیم‌ها بازی کند، با این شرط که هر تیم در هر روز بیش از یک بازی انجام ندهد.

۳.۶ روش تخصیم و حل برای طراحی الگوریتمها

باید به پرسش‌های زیر پاسخ دهیم:

- الف) حداقل در چند روز بازی‌ها قابل انجام است؟
ب) برنامه‌ی زمان‌بندی بازی‌ها چیست؟

تعداد کل بازی‌ها برابر $1 - \frac{1}{n}$ است و در هر روز حداکثر $\lceil \frac{1}{n} \rceil$ تعداد بازی می‌تواند انجام شود. پس حد پایین تعداد روزهای دوره‌ی بازی اگر n روز باشد برابر $1 - \frac{1}{n}$ و اگر n فرد باشد برابر $n - \frac{1}{n}$ روز است. پس برنامه‌ی زمان‌بندی بازی‌ها برای n نیم در m روز را می‌توان با یک ماتریس A به ابعاد $m \times n$ مدل کرد.

درایه‌ی $A[i,j] = k$ یعنی نیم i با k در روز j بازی می‌کند. در این ماتریس هم‌جنین،

۱. در سطر نام اعداد $1 \dots n$ به جزء قرار دارند.

۲. در هر چهار سطر و ستونی عدد تکراری نداریم.

۳. رابطه‌ی زیر صادق است:

$$A[i,j] = a \Leftrightarrow A[a,i] = j$$

ایندا مسئله را برای $2^k = n$ حل می‌کنیم. ادعای می‌توان بازی‌ها را در $1 - \frac{1}{n}$ روز انجام داد. این ادعا را با استفرا اثبات می‌کنیم.

برای دو نیم واضح است که در یک روز بازی انجام می‌شود. این نیم را به دو گروه 2^{k-1} نسبی نسبی می‌کنیم. طبق فرض استفرا، هر گروه می‌تواند بازی‌های بین خود را در $1 - \frac{1}{n}$ روز انجام دهد. حال فقط بازی‌های هر نیم از گروه دوم با نیمه‌ای گروه اول (و عکس) باقی مانده است. از این بازی‌ها می‌توان برنامه‌ی بازی‌های هر نیم از گروه اول را هم به دست آورد. مدل ماتریسی بازی‌های ساقی مانده مطابق شکل ۱۸.۴ است. اگر محوایم کل بازی‌ها را در $1 - \frac{1}{n}$ روز انجام دهیم، این ماتریس باید پس ماتریس مرتعی به ابعاد $n \times n - \frac{1}{n}$ باشد به طوری که در آن تنها اعداد $1 \dots n$ آمده باشد. همچنین در هر سطر و در هر ستون این ماتریس عدد تکراری نباید داشته باشیم. این جنبن ماتریسی به «مرتع لاتین^{۱۱}» مشهور است و برای تولید آن الگوریتم‌های متعددی وجود دارد. پس از این راه حل‌ها این است: اعداد $1 \dots n$ را در سطر اول می‌نویسیم و اعداد هر سطر بعدی را از با دوران به راست (با چپ) سطر قبلی اش به اندازه‌ی یک خانه به دست می‌آوریم. به مثال شکل ۱۸.۴ ب) مراجعه کنید. از این مرتع لاتین، مرتع لاتین بالای آن (برنامه‌ی بازی‌های نیم‌های گروه اول با دوم هم به دست می‌آید.

به این صورت، کل بازی‌ها در $1 - \frac{1}{n} + \frac{1}{n} = 1 - \frac{1}{2}$ روز انجام می‌شود.

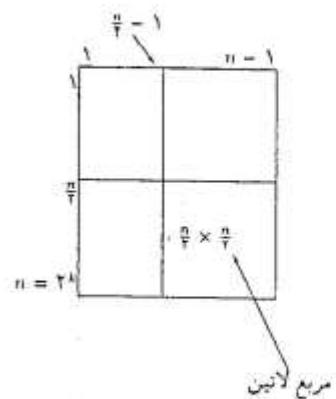
حال اگر n نوانی از دو نباشد. اگر n فرد بود، با اضافه کردن یک نیم «اضافی» حل مسئله مانند حالت زوج می‌شود و یک روز به زمان بازی‌ها اضافه می‌گردد. در زمان‌بندی‌ای که به دست می‌آید، هر نیم که با نیم «اضافی» بازی دارد، در واقع استراحت می‌کند.

برای حل در حالت کلی هم از استفرا استفاده می‌کنیم. فرض می‌کنیم حل مسئله را برای $2^k = n$ نیم داریم، دو حالت وجود دارد:

الف) $n/2$ روز است. در این صورت دقیقاً مانند حالت $2^k = n$ عمل می‌کنیم. هر گروه بازی‌های خودش را در $1 - \frac{1}{2}$ روز انجام می‌دهد و برنامه‌ی بازی‌های بین دو گروه از طریق مرتع لاتین به دست می‌آید.

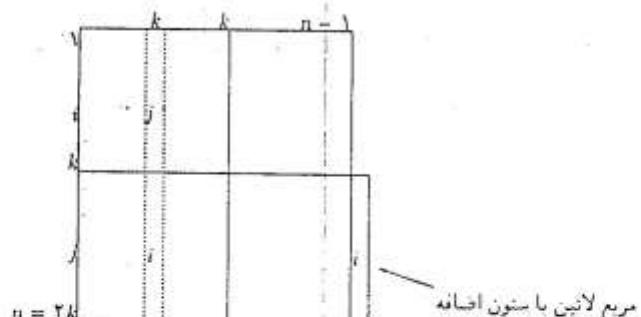
latin square^{۱۲}

۲	۳	۴	۵	۶	۷	۸
۱	۴	۲	۸	۵	۶	۷
۴	۱	۲	۷	۸	۵	۶
۳	۲	۱	۳	۷	۸	۵
۶	۷	۸	۱	۲	۳	۴
۵	۸	۷	۴	۱	۲	۳
۸	۵	۶	۳	۴	۱	۲
۷	۶	۵	۲	۳	۴	۱



(ب) مثال برای $n = 4$

شکل ۱۸.۴: زمان‌بندی دوره‌ی باری‌ها برای $2^k = n$ قسم.



شکل ۱۹.۴: حالی که $n/2$ فرد است.

۳.۴ روش تقسیم و حل برای طراحی الگوریتمها

۱	۲	۳
۲	۱	۴
۳	۴	۱
۴	۳	۲

۱
۲

شکل ۳.۴: برنامه‌ی بازی‌ها برای $n = 2$ و $n = 4$

۱	۲	۳	۴	۵
۲	۱	④	۶	۵
۳	⑦	۱	۲	۵
۴	۵	۶	①	۲
۵	۴	⑦	۶	۲
۶	⑦	۴	۵	۱

۱	۲	۳
۲	-	۲
-	۱	۲

شکل ۳.۵: برنامه‌ی بازی‌ها برای $n = 3$ و $n = 6$

ب) $n/2$ فرد است. در این صورت هر گروه بازی‌های بین خودشان را در $n/2$ روز انجام می‌دهند. نتیجه به این که در هر روز از این $n/2$ روز (طبق استغفار) یک تیم از هر گروه قرار است استراحت کند، به جای استراحت این تیم‌ها (کی از گروه اول و دیگری از گروه دوم) می‌توانند با هم بازی کنند.

با این ترتیب، تعداد روزهای باقی مانده برای بازی‌تیم‌های گروه اول با گروه دوم $1 - n/2$ است. مجدداً ماتریس C به ابعاد $(1 - n/2) \times n/2$ برای بازی‌های تیم‌های گروه دوم با گروه اول را در نظر بگیرید (شکل ۳.۶). این ماتریس باید حاری اعداد 1 تا $n/2$ را باشد با این شرط که در هر سطر و ستون آن اعداد تکراری نداشته باشیم. هم‌چنین برای هر تیم i از گروه دوم، می‌دانیم که این تیم در $n/2$ روز اهل بازی‌ها، دقیقاً یک تیم زی از گروه اول بازی کرده است. این به معنی آن است که در سطر iام ماتریس C باید همه‌ی اعداد 1 تا $n/2$ باید به جزء

برای اعمال آخرین شرط فوق برای ماتریس C، یک ستون به آن اضافه می‌کنیم تا مربع شود و در سطر iام این ستون عدد α می‌نویسیم (α و زدن بالا تعریف شده‌اند). حال ماتریس C با ستون اضافی به یک مربع لاتین تبدیل می‌شود که یکی از ستون‌های آن از قابل مشخص است. اگر این مربع را 90° درجه بچرخانیم می‌توانیم همان الگوریتم تولید مربع لاتین، که در بالا گفته شد، با ان تفاوت که در سطر اول جای گشتنی از اعداد 1 تا

۱	۲	۳	۴	۵	۶	۷	۸	۹
۲	۳	۴	۵	۶	۷	۸	۹	۱۰
۳	۱	۲	۳	۴	۵	۶	۷	۸
۴	۵	۶	۷	۸	۹	۱۰	۱	۲
۵	۶	۷	۸	۹	۱۰	۱	۲	۳
۶	۷	۸	۹	۱۰	۱	۲	۳	۴
۷	۸	۹	۱۰	۱	۲	۳	۴	۵
۸	۹	۱۰	۱	۲	۳	۴	۵	۶
۹	۱۰	۱	۲	۳	۴	۵	۶	۷
۱۰	۱	۲	۳	۴	۵	۶	۷	۸

۲	۳	۴	۵
۱	۰	۲	۴
۳	۱	۲	۰
۰	۲	۱	۴
۵	۲	۳	۱

شکل ۲۲.۴: برنامه‌ی بازی هارای $n = 10$ برای $n = 5$.

۲/۱ قرار دارد، را اعمال کنیم و ماتریس C را به دست آوریم. مانند قبل، قسمت چهارم ماتریس کلی را می‌توانیم از C تولید کنیم.

در این مثال برای $n = 10$ آورده شده است. ابتدا باید مسئله را برای $n = 5$ حل کنیم و برای آن باید ماتریس را برای $n = 5$ به دست آوریم.

$$10/2 = 5 \Rightarrow 5 + = 6 \Rightarrow 6/2 = 3 \Rightarrow 3 + = 4 \Rightarrow 4/2 = 2$$

پس ابتدا برای $2 = n$ و سپس برای $4 = n$ مسئله را حل می‌کنیم (شکل ۲۱.۴).

آن‌گاه برای $2 = n$ و سپس برای $6 = n$ مسئله را حل می‌کنیم (شکل ۲۱.۴).

و در نهایت، مسئله را برای $5 = n$ و سپس برای $10 = n$ حل می‌کنیم (شکل ۲۲.۴).

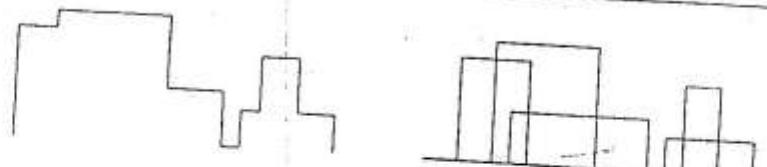
۲.۳.۴ مسئله‌ی برج‌ها

«برج داده شده‌اند، همه‌ی برج‌ها بر روی محور یک‌جا قرار دارند و برج آبایه عدد a_i, b_i, c_i مشخص می‌شود، که در آن $i = 1, 2, \dots, n$ و آبایه ترتیب مختصات نقطه‌ی چپ و پایین، راست و پایین و ارتفاع برج نام است. می‌خواهیم «نمای برج‌ها»، آن چه از دور دیده می‌شود را به دست بیاوریم. نمای برج‌ها، لبه‌ی خارجی حاصل از کنار هم قرار گرفتن برج‌هایست که دیده می‌شود.

بگ نمای را می‌توان به صورت دنباله‌ی $< h_{1,1}, h_{1,2}, \dots, h_{1,n}, h_{2,1}, \dots, h_{2,n}, \dots, h_{n,1}, h_{n,2}, \dots, h_{n,n} >$ نمایش داد که در آن دنباله مختصات گوشه‌های تعدادی از برج‌هایست (همه‌ی آنها) که دیده می‌شوند. این آنها مرتب می‌باشند ($< h_{1,1}, h_{1,2}, \dots, h_{1,n}, h_{2,1}, \dots, h_{2,n}, \dots, h_{n,1}, h_{n,2}, \dots, h_{n,n} >$). همچنان ارتفاع نمایین x_1, x_2, \dots, x_n برای n است و ارتفاع نمای برای $x_1 < x_2 < \dots < x_n$ صفر است. برای مثال شکل ۲۲ نمای چند برج را نشان می‌دهد.

۲۳ روش تقسیم و حل برای طراحی الگوریتمها

۱۱۱



شکل ۲۳.۴: (الف) آسمان خراش‌ها و (ب) نمای آن‌ها

راه حل ۱ - برای $n = 1$ زندگاله همان سه عدد تهای برج است.
برای $n > 1$ به صورت استقرآ حل می‌کنیم و دنباله‌ی آن را به دست می‌آوریم. حال ساختمان S ام را با مشخصات a_n, b_n, c_n به این شما اضافه می‌کنیم.
برای این کار کافی است z_i و z_{i+1} را در نمای پیدا کنیم که $z_i \leq z_j \leq z_{i+1}$ (حالت نساوی،
حال خاص است). حال اگر $c_n < h_i, z_{i+1} < a_n, c_n >$ را پس از z_i در S درج می‌کنیم. هم‌چنین اگر
 $c_n < h_i, z_{i+1} < b_n, c_n >$ را پس از z_i در S درج می‌کنیم. حال ارتفاع‌های $a_1, b_1, c_1, \dots, a_n, b_n, c_n$
را به ترتیب با c_n مقایسه می‌کنیم و اگر برای $j < i$ بود، در S $h_j < c_n < z_{i+1}$ قرار می‌دهیم. تنها
مرحله‌ای که باقی مانده است حذف نقاط متواالی همارتفاع در S است، که با به راحتی قابل انجام است (البته این کار
را در همان پوش اول هم می‌توان انجام داد).
برای ساده‌تر شدن پیاده‌سازی توقیع بصر است به اول زندگاله S زوج $> -\infty$ و به آخر زندگاله زوج
 $> +\infty$ را اضافه کنیم.

پیچیدگی الگوریتم
پیداکردن a_n و b_n در زندگاله S ، با $\Theta(\log k)$ امکان‌پذیر است. ولی تصحیح ارتفاع‌ها می‌تواند حد اکثر $\Theta(k)$ باشد.
چون $n \leq k$ داریم:

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

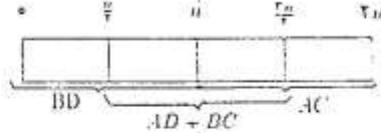
راه حل ۲ - برای این مسئله از طریق تقسیم و حل راه حل کارانتری وجود دارد. ایندا مسئله‌ی باندازه‌ی n را
به دو مسئله با اندازه‌های $\lceil n/2 \rceil$ و $\lfloor n/2 \rfloor$ تقسیم می‌کنیم و هر دو زیرمسئله را به همین روش حل می‌کنیم.
فرض کنید تمامی زیرمسئله‌ی اول $S_1 = \langle z_1, h_1, z_2, h_2, \dots, z_k, h_k, z_{k+1}, \dots, z_{k+1}, h_{k+1}, z_{k+2}, h_{k+2}, \dots, z_{k+2}, h_{k+2} \rangle$ و تمامی زیرمسئله‌ی دوم
برای به دست آوردن نمای مسئله پاید $S_2 = \langle z_r, h_r, z_{r+1}, h_{r+1}, \dots, z_p, h_p, z_{p+1}, \dots, z_q, h_q \rangle$ باشد.
آرایه‌ی مرتب در MergeSort است: بین دو زندگاله، کوچکترین z را انتخاب می‌کنیم و آن را همراه با ارتفاع بعدیش
در زندگاله جواب می‌رسیم. حال اعمال زیر را با پایان یافتن دنباله‌ها نکاریم: سراغ بعدی می‌رویم (بین دو
ز کوئی دو زندگاله، z کوچک‌تر را انتخاب می‌کنیم) و اگر ارتفاع آن بیشتر از آخرین ارتفاع نوشته شده در زندگاله
جدید است، آن را به همراه بعدیش در زندگاله جواب می‌نویسیم.

پیچیدگی الگوریتم: فرمت مهم این الگوریتم ادغام دو زندگاله است که مانند ادغام دو آرایه‌ی مرتب از مرتبه‌ی

$\Theta(\lceil \frac{n}{2} \rceil + \lfloor \frac{n}{2} \rfloor) = n$ است. پس

$$T(n) = 2T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor) + \Theta(n) \Rightarrow T(n) = \Theta(n \log n)$$

۱۱۱



سکل ۴.۴: مدل ساری مسئله ضرب دو چندجمله‌ای.

که سریع تر از قابلی می‌باشد.

می‌توانیم استدلال کنیم که حد پایین این مسئله $\log n$ است. اگر دستاله ورودی شامل برج‌هایی باشد که هیچ دویابی از آن‌ها بهم نداخل نداشته باشند با حل کردن مسئله برج‌ها دنباله‌ای مرتب شده از بینها به دست می‌آید، اگر این عمل را در زمانی بهتر از $\log n$ انجام دهیم، به الگوریتم مرتب‌کننده‌ی مبتنی بر مقایسه‌ای رسیده‌ایم که در زمانی بزرگتر از $n \log n$ مرتب می‌کند. این غیرممکن است!

۴.۳.۴ ضرب دو چندجمله‌ای

دو چندجمله‌ای P و Q بر حسب x و از درجه‌ی n داده شده‌اند:

$$P_n(x) = a_0x^n + a_{n-1}x^{n-1} + \dots + a_1$$

$$Q_n(x) = b_0x^n + b_{n-1}x^{n-1} + \dots + b_1$$

می‌خواهیم با یک الگوریتم کارا حاصل ضرب $R_{2n}(x) = P_n(x)Q_n(x) = \sum_{i=0}^{2n} r_i x^i$ را محاسبه کنیم. پس در این مسئله،

ورودی: $a_0, a_{n-1}, \dots, a_1, b_0, b_{n-1}, \dots, b_1$

خروجی: r_0, r_{n-1}, \dots, r_n

برای پیاده‌سازی می‌توانیم ورودی را دو آرایه‌ی A و B و D را و خروجی را در آرایه‌ی C $2n+1$ عنصری سوییم (سکل ۴.۴). راه حل ۱ - برای $n \leq k \leq 2n$ داریم، $r_k = \sum_{i+j=k} a_i b_j$ به این ترتیب، ها را می‌توان با $O(n^3)$ به دست آورد.

```

for i:=0 to 2*n do      C[i]:=0;
for i:=0 to n do
  for j:=0 to n do
    C[i+j]:=C[i+j]+A[i]*B[j];
  
```

راحل ۲ - ما با استفاده از تقسیم و حل، می‌کنیم روشی کارانه‌ای برای حل این مسئله بیاییم. در اینجا فرض می‌کنیم $n = 2^k$. هر چند الگوریتم در حالت کلی هم درست است.

۳.۶ روش تقسیم و حل برای طراحی الگوریتم‌ها

در ابتداء P و Q را به دو نیمه تقسیم می‌کنیم:

$$P_n(x) = A_1 x^{\frac{n}{2}} + B_1$$

$$Q_n(x) = C_2 x^{\frac{n}{2}} + D_2$$

که در آن A_1, B_1, C_2 و D_2 چندجمله‌ای‌هایی از درجه‌ی $\frac{n}{2}$ هستند. حالا حاصل ضرب PQ برابر است با:

$$PQ(x) = A_1 C_2 x^n + A_1 D_2 + B_1 C_2 x^{\frac{n}{2}} + B_1 D_2$$

در این روش برای به دست آوردن PQ باید چهار بار و هر بار دو چندجمله‌ای از درجه‌ی $\frac{n}{2}$ را به صورت بارگذشت در هم ضرب کنیم و ضرایب حاصل را با توجه به میزان «شیفت» که از ضرب در $\frac{n}{2}$ به دست می‌آید باهم جمع نماییم، یه عبارت دیگر، $C_2 \cdot C_2$ ، یعنی به دست آوردن بردار ضرایب چندجمله‌ای حاصل از ضرب A_1 و B_1 . جمع این ضرایب با مقادیر موجود در آرایه C پس از اعمال «شیفت» برای تعابیل این الگوریتم داریم:

$$T(n) = 4T\left(\frac{n}{2}\right) + \Theta(n)$$

که در نتیجه با استفاده از قضیه اصلی $T(n) = \Theta(n^k)$ است. پس الگوریتم کار نمی‌نماید.

راه حل ۲ - ما می‌توانیم این روش را بهتر کنیم. به اتحاد ربرو توجه کنید:

$$AD + BC = (A - B)(D - C) + AC + BD$$

اگر به جای محاسبه $AD + BC$ از اتحاد بالا استفاده کنیم، می‌توانیم حاصل ضرب‌های BD و AC را ذخیره کنیم و در جای دیگر از آن استفاده کنیم. در نتیجه فقط نیاز به ۳ بار ضرب دو تا چندجمله‌ای از درجه‌ی $\frac{n}{2}$ داریم. در این صورت خواهیم داشت:

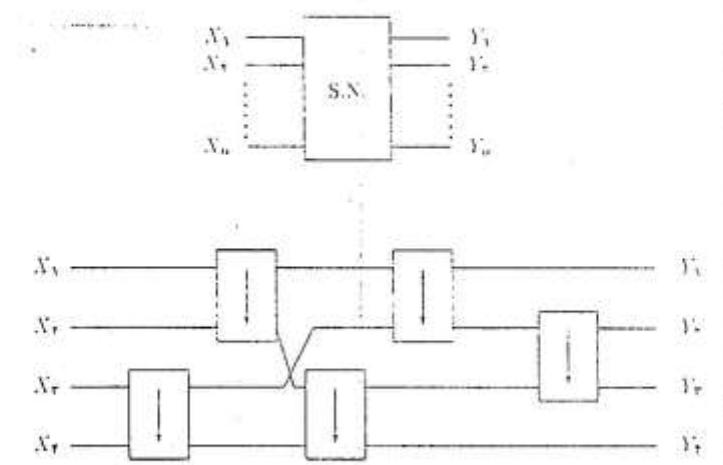
$$T(n) = 3T\left(\frac{n}{2}\right) + \Theta(n)$$

و با استفاده از قضیه اصلی داریم: $T(n) = \Theta(n^{\log_2 3})$. نتیجه این کارایی در n های بالا مشهود است. این تکنیک را آفای استراتژیون^{۲۲} برای ضرب دو ماتریس هم به کار برده است. ضرب دو ماتریس در کارهای مهندسی کاربرد زیادی دارد. در روش عادی از مرتبه‌ی n^4 است. اما با روش استراتژیون به درجه‌ی $n^{\log_2 3}$ کاهش پائیه است. و اخیرا نیز به ^{۲۳} "رسیده" است.

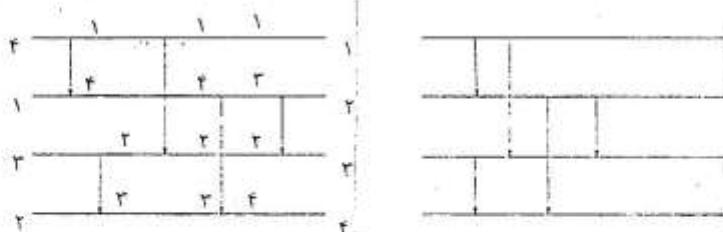
۵.۳.۴ شبکه‌های مرتب ساز

می‌خواهیم با استفاده از مقایسه کننده‌ها به عنوان المان اصلی، مداری ساریم تا حداقل تعداد مقایسه کننده‌ها و پرحتی الامکان با حداقل تأخیر^{۲۴} عدد چند بیشی را مرتب نماییم. چرا مدار شکل ۲۶.۲ درست کارمی کند؟ آیا باید اینمی‌باشد؟ جایگشت را بررسی کنیم؟ مکل ۲۷.۴ مداریک مقایسه کننده‌ی دوتایی را نشان می‌دهد.

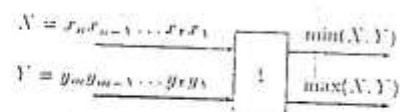
STRASSEN^{۲۵}



شکل ۲۵.۲: مقایسه کلی پک شبکه های مرتب ساز

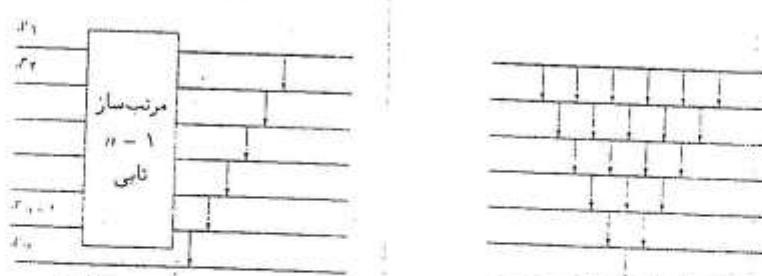
شکل ۲۶.۴: دو شبکه های مرتب ساز برای $n=4$ مقایسه کننده ها با علامت برشان داره شده اند.

۳.۴ روش تقسیم‌ریختی طراحی الگوریتم‌ها

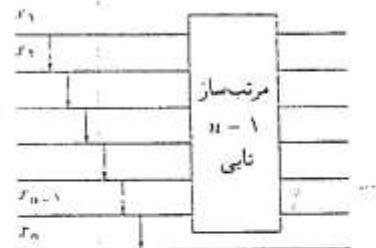


Current State	input	Next State	output
0	00	0	00
0	01	1	01
0	10	2	01
0	11	0	11
1	x y	1	x y
2	x y	2	y x

شکل ۲۷.۴: مدار مقایسه‌کننده دوتایی.



شکل ۲۸.۴: مدار مقایسه‌کننده مبنی بر درج.



شکل ۲۹.۴: مدار مقایسه‌کننده‌ی مبتنی بر انتخاب.

روشهای مختلف طراحی شبکه‌های مرتب‌ساز

الف- مبتنی بر درج (مانند insertion sort)، مانند شکل ۲۸.۴
تحلیل:

$S(n)$: اندازه‌ی شبکه (تعداد مقایسه‌کننده‌ها)

$T(n)$: زمان تاخیر مدار؛ یعنی حداقل تعداد مقایسه‌کننده‌ای که یک عدد ورودی باید از آن عبور کند تا به حروفی برسد. برای این مدار داریم:

$$S(n) = \begin{cases} 0 & n = 1 \\ S(n-1) + n - 1 & n > 1 \end{cases}$$

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

$$T(n) = \begin{cases} 0 & n = 1 \\ T(n-1) + 2 & n > 1 \end{cases} \Rightarrow T(n) = 2n - 2$$

ب- مبتنی بر انتخاب (مانند bubble sort) مانند شکل ۲۰.۴

اندازه و زمان این شبکه هم مانند شبکه قلی است.

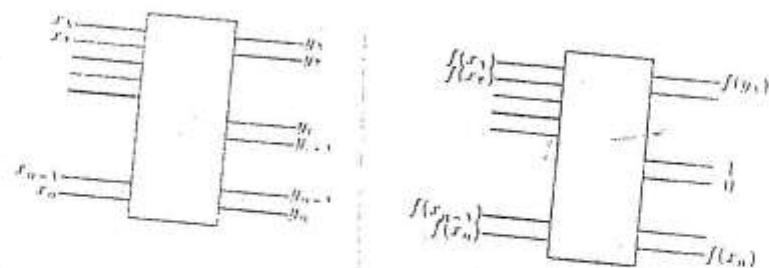
این اثبات درستی مدارهای قلی ساده است؛ ولی برای مدارهای دیگر جظر!

قضیه‌ی صفر و یک (Zero and One Principle)

قضیه ۹. شرط لازم و کافی برای این که یک شبکه‌ی مرتب‌ساز درست کار کند این است که گله‌ی ترتیب‌های «صفر و یک را به صورت غیر نزولی مرتب کند».

این اثبات شرط لازم بدینهی است، برای شرط کافی، ورض کنید که $f(x) \leq f(y)$ اگر $y \leq x$. در آن صورت اگر شبکه‌ای جای گشت (x_1, \dots, x_n) را به (y_1, \dots, y_n) تبدیل کند.

۳.۴ روش تقسیم و حل برای طراحی الگوریتمها



شکل ۳.۴: اثبات قضیهی صفر و یک.

آنکه $(f(x_1), \dots, f(x_n))$ را بزیر به $((f(y_1), \dots, f(y_n))$ تبدیل می‌نماید (شکل ۳.۴). حال با استفاده از برهان خلف فرض کنید که برای یک n , $y_{n+1} > y_i$ است. در این صورت نایم را به صورت زیر انتخاب می‌کنیم:

$$f(x) = \begin{cases} 0 & x < y_i \\ 1 & x \geq y_i \end{cases}$$

در این صورت در خروجی ۱ جلوی خواهد بود که مخالف فرض قضیه است.

ترکیب‌ساز زوج‌فرد (Odd-Even Merger)

شکل ۳.۴.۲ نحوی ساختن این شبکه را نشان می‌دهد. فرض بر این است که a_1, \dots, a_n و b_1, \dots, b_n به صورت غیر نزولی مرتب هستند

قضیه ۱۰. اگر $a_1, \dots, a_n = \vec{a}$ و $b_1, \dots, b_n = \vec{b}$ به صورت غیر نزولی مرتب باشند، $c_1, \dots, c_n = \vec{c}$ در شبکه

شکل ۳.۴.۳ به صورت غیر نزولی مرتب خواهد بود.

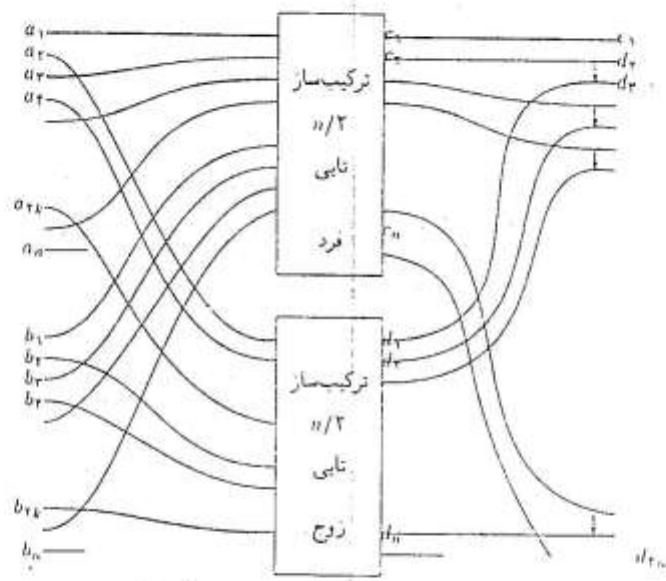
اثبات. با استفاده از قضیهی صفر و یک، فرض می‌کنیم که \vec{a} و \vec{b} به صورت های زیرند:

$$\vec{a} = a_1, \dots, a_n = \underbrace{0 \dots 0}_{n-f}, \underbrace{1 \dots 1}_{f}, \dots, 1$$

$$\vec{b} = b_1, \dots, b_n = \underbrace{0 \dots 0}_{n-y}, \underbrace{1 \dots 1}_{y}, \dots, 1$$

با استفاده از فرض استقرار، داریم:

$$\vec{c} = \underbrace{0 \dots 0}_{n-f+y}, \underbrace{1 \dots 1}_{f-y}, \dots, 1$$



شکل ۴.۲۱.۴: ترکیب ساز زدج فرد.

۳۰۴ روش تفہیم و حل برای طراحی الگوریتم‌ها

$$\vec{d} = \underbrace{\circ \circ \dots \circ}_{n} \underbrace{\backslash \backslash \dots \backslash \backslash}_{m}$$

بس احتلای تعداد صفرهای ۰ و ۱ برآورده است: $((1/2) + (1/2)) + ((1/2) + (1/2))$ و این مقدار برآورده ۱.۰ و با ۲ است. این سه حالت را بررسی می‌کیم:

۷. تعداد صفرهای ۰ و آن برابر هستند:

$$\left\{ \begin{array}{l} \overline{r} = 000\ldots00111\ldots1 \\ \overline{l} = 000\ldots00111\ldots1 \end{array} \right\} \Rightarrow \overline{e}$$

^۱ اختلاف تعداد صفرهای آن و آن را برابر ۱ است:

$$\left. \begin{array}{l} \vec{c} = 000\dots00111\dots1 \\ \vec{d} = 000\dots01111\dots1 \end{array} \right\} \Rightarrow \text{مرتب است } \vec{e}$$

۲. اختلاف تعداد صفرهای ۰ و آنرا ۲ است:

$$\left. \begin{array}{l} \vec{c} = \langle \dots, 0, 0, 1, 1, 1, \dots \rangle \\ \vec{d} = \langle \dots, 0, 1, 1, 1, 1, \dots \rangle \end{array} \right\} \text{مرتب است} \Rightarrow \vec{e}$$

پس در هر حالت آن مرتب است.

مرتبه ساز روح و فرد

این مرتب ساز بر اساس ترکیب ساز روح فرد است (نکل ۴.۲۲).
آنات درستی واپسی است.

حلیل ترکیب‌ساز و مرتضی‌خانی

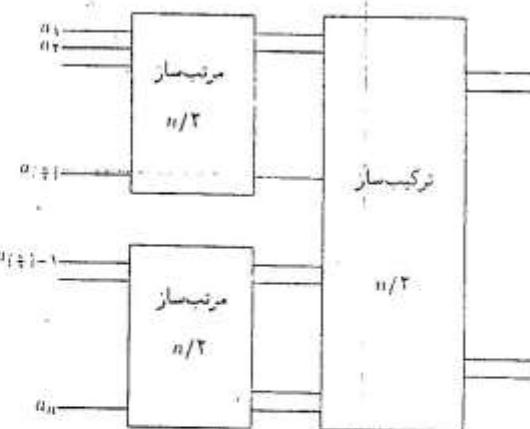
M(n) : الاداري ترکیبیان

۳-۱۰۵: انداره‌ی متسا-ویار

m(n): محقق فریکیپیا (همان، همان است).

سازمان اسناد و کتابخانه ملی

$$M(n) = \begin{cases} n & n = \lambda \\ \gamma M(n/\gamma) + n - \lambda & \text{otherwise} \end{cases} \implies M(n) = n \log_{\gamma} n + \lambda$$



شکل ۴: مرتب‌ساز زوج فرد.

$$S(n) = \tau S(n/\tau) + M(n/\tau) = \frac{n}{\tau} \log^{\tau} n + O(n)$$

$$m(n) = \begin{cases} 1 & n = 1 \\ m(n/\tau) + 1 & n > 1 \end{cases} \implies m(n) = \log n + 1$$

$$s(n) = s(n/\tau) + m(n/\tau) = \frac{1}{\tau} \log n (\log n + 1)$$

بدینهی است که حد بازیس برای اندازه و عمق (زمان) هر شبکه‌ی مرتب‌ساز به ترتیب $\Theta(\log n)$ و $\Theta(n \log n)$ است (حرا?

در سال ۱۹۸۲ Szemardí و Komlosi Ajtai موفق به ارایه‌ی شکه بهینه‌ای با اندازه‌ی $O(n \log n)$ و عمق $O(\log n)$ نمودند.^{۴۴} این شکه به نام AKS معروف است.

^{۴۴} آنلاین نات پیار بزرگ: [http://www.cs.cmu.edu/~rbarzani/paper/aksm.pdf](#)

۴.۴ روش برنامه‌ریزی پویا

مسئله‌هایی هستند که حل آن‌ها به روش تقسیم‌وحل موجب حل نکاری زیرمسئله‌های از این مسئله می‌شود. این گونه مسئله‌ها را بده است به جای حل از بالا بهایین و به صورت بازگشتی، اول پایین به بالا حل کنیم؛ زیرمسئله‌ها فقط یکبار حل کرد، و حاصل آن‌ها را در جدولی ذخیره کنیم، تا اگر برای حل زیرمسئله‌های بزرگ‌تر مکرراً به تبیه‌ی حل یک ریزمسئله‌ی کوچک‌تر نیاز باشد، آن را بنوان بدون برداخت هزینه‌ای از جدول به دست آورد. این روش حل را «برنامه‌ریزی پویا»^{۳۲} می‌گوییم. در مورد ویژگی‌های مسئله‌هایی که راه حل پویا دارند، در ادامه‌ی این بخش بیشتر صحبت خواهیم کرد. ولی قبل از آن به یک مسئله این چنینی توجه کنید.

۱.۴.۴ ترکیب m از n

می‌خواهیم با دریافت n و m ، حاصل $\binom{n}{m}$ را نهایا عمل جمع انجام دهیم. برای این کار به تعریف ریز توجه کنید:

$$\binom{n}{m} = \begin{cases} 1 & n = m \\ \binom{n-1}{m} + \binom{n-1}{m-1} & \text{در غیر این صورت} \end{cases}$$

اگر این مسئله را از طریق تقسیم‌وحل، حل کنیم خواهیم داشت:

```
function Combination (n,m:integer):integer;
begin
  if (n=m) or (m=0) then return (1)
  else
    return (Combination(n-1,m)+Combination (n-1,m-1));
end;
```

اگر $T(n,m)$ تعداد اعمال جمع این پردازه باشد، داریم:

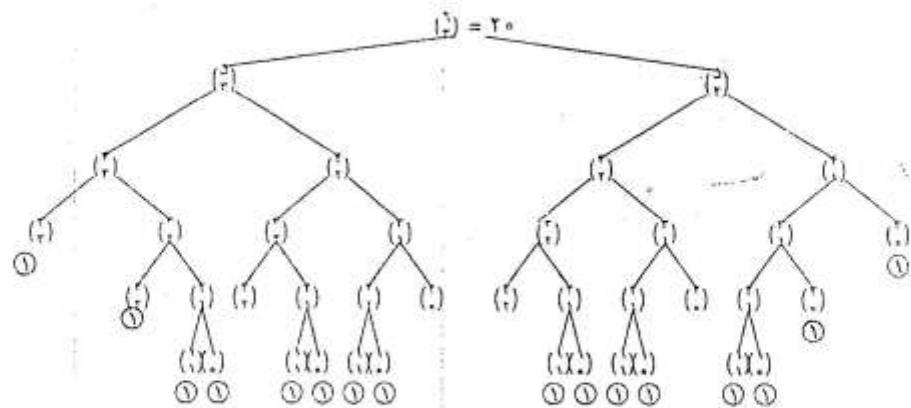
$$T(n,m) = \begin{cases} 0 & n = m \\ T(n-1,m) + T(n-1,m-1) + 1 & \text{در غیر این صورت} \end{cases}$$

با استفاده از فرمول اصلی می‌توان دید که $1 - \binom{n}{m} = T(n,m)$. لذت این جواب را از راه سریع‌تری هم می‌توان به دست آورد و آن این است که $\binom{n}{m}$ از راه فوق فقط از جمع یک‌ها به دست می‌آید. بنابراین، تعداد جمع‌ها $1 - \binom{n}{m}$ است.

این الگوریتم هر جند ساده است، ولی یک انکال مهم دارد. انکال الگوریتم این است که زیرمسئله‌های زیادی را چندین بار به طور مستقل حل می‌کند و حاصل آن را به دست می‌آورد. مثلًا برای محاسبه $\binom{10}{5}$ بار محاسبه می‌شود.

روش بعدتری برای حل این مسئله وجود دارد و آن این است که زیرمسئله‌ها فقط یک بار حل شوند و حاصل آن‌ها در جدولی برای استفاده مجدد ذخیره شوند. این روش حل، همان روش برنامه‌ریزی پویا است.

dynamic programming

شکل ۳۲.۴: درخت بازگشت برای به دست آوردن $C(i,j)$

در این روش یک ماتریس به ابعاد $(n+1) \times (m+1)$ به نام C در نظر می‌گیریم که $C(i,j)$ ، $0 \leq i \leq n$ و $0 \leq j \leq m$ ، حاوی مقدار $C(i,j)$ خواهد بود. بدینهی است که قطر و ستون صفر این ماتریس برابر ۱ است. و درایهی $C(i,j)$ برابر است با مجموع درایه‌های $(i-1, j-1)$ و $(i, j-1)$. الگوریتم پیدا کردن $C(n,m)$ به صورت زیر درمی‌آید:

	۰	۱	۲	۳	۴	۵	۶
۰	۱						
۱	۱	۱					
۲	۱	۲	۱				
۳	۱	۳	۲	۱			
۴	۱	۴	۶	۴	۱		
۵	۱	۵	۹	۱۰	۵	۱	
۶	۱	۶	۱۵	۱۹	۱۰	۷	۱
۷	۱	۷	۲۱	۲۴	۲۴	۲۱	۷
۸	۱	۸	۲۸	۳۵	۳۸	۳۵	۲۸
۹	۱	۹	۳۶	۴۲	۴۰	۴۲	۴۵
۱۰	۱	۱۰	۴۵	۴۹	۴۶	۴۶	۴۹

شکل ۳۴.۴: ماتریس برای محاسبه‌ی $C_{10,10}$

```

Function Combination( n,m :integer):integer;
var i,j:integer;
C: array [0..100,0..100] of integer;
begin
  for i:=1 to n do C[i,0]:=1;
  for i:=1 to m do C[i,i]:=1;
  for i:=2 to n do
    for j:= 1 to min(i-1, m) DO
      C[i,j]:= C[i-1,j] + C[i-1,j-1];
  return( C[n,m] );
end;

```

عناصر سطرهای این ماتریس در حقیقت همان مثلث خیام (پاسکال) هستند. شکل ۴.۲، ماتریس نهایی برای محاسبه $C(n,m)$ را نشان می‌دهد.
نوجه کنید که در این راه حل کافی است به جای ماتریس از یک آرایه $A[0..m]$ که پک سطر ماتریس را در خود دارد استفاده کنیم. البته در این صورت حلقه‌ی داخلی را باید از اندیس بالا به پایین برگردان، یعنی

```

Function Combination( n,m :integer):integer;
var i,j:integer;
A: array [0..100] of integer;
begin
  A[0]:=1;
  for i:=2 to n do
  Begin
    if i<=m then A[i]:=1;
    for j:= min(i-1, m) downto 1 DO
      A[j]:= A[j] + A[j-1];
  End;
  return(A[m]);
end;

```

تعداد اعمال جمع در این الگوریتم برآورد

$$T(n,m) = 1 + 2 + \dots + m - 1 + m(n-m) = m(n-m+1)/2$$

است که برای برخی مقادیر n و m کوچکتر از $(n-m)$ ولی برای مقادیری مانند $n = m$ بزرگتر از آن است، علت آن محاسبه‌ی درایه‌های از ماتریس است که نشان در متدار $C(n,m)$ ندارند. در واقع محاسبه‌ی درایه‌هایی که در نکل با x نشان داده شده‌اند کفایت می‌کند.

	0 1 2 3 4 ... n
0	1
1	1 1
2	1 x 1
3	1 x x 1
4	x x x 1
.	x x x 1
.	x x x 1
.	x x x 1
.	x x x 1
.	x x x
n	x

۱) حلقه‌ی داخلی الگوریتم با این تغییرات صورت زیر در خواهد آمد.

```

Begin
    if i<=m then A[i]:=1;
    for j:= min(i-1, m) downto minimum(1,m-n+i) Do
        A[j]:= A[j] + A[j-1];
End;

```

تعداد اعمال جمع در این صورت برابر است با $(n-m)m$ که با توجه به این که $\binom{n}{n-m} = \binom{n}{m}$ این تعداد حداقل است و داریم $1 - \binom{n}{m} \leq (n-m)m$.

مسایلی که به روش برنامه‌برزی پویا حل می‌شوند، دارای وزنگی‌های زیر هستند.

۱. مساله بهینه‌سازی است. (پارامتری کمینه یا بیشینه شود)

۲. زیر مسائل آن هم بهینه است.

۳. حل مسئله به روش تقسیم و حل، موجب تکرار حل زیر مسئله‌های منابع خواهد شد. بعد از داشتن وزنگی

اول و دوم چک می‌کنیم آیا زیر مسائل به صورت تکراری حل می‌شوند یا نه؟ اگر این گونه بود از این روش

استفاده می‌کیم.

۲.۴.۴ ضرب ماتریس‌ها

ماتریس M_1 تا M_n داده شده‌اند. ابعاد ماتریس M_i برابر $d_{i-1} \times d_i$ است. می‌خواهیم این ماتریس‌ها را به صورت زیر در هم ضرب کیم:

$$M = M_1 \times M_2 \times \dots \times M_n$$

۴.۴ روش برنامه‌ریزی پریا

ترتیب انجام اعمال حرب را طوری تعیین کنید تا تعداد کل ضرب‌های اعداد حقیقی کمینه شود. می‌دانیم که برای ضرب در ماتریس $A \times B$, که ابعاد A و B به ترتیب $p \times r$ و $r \times q$ هستند به تعداد pqr ضرب اعداد حقیقی انجام می‌شود.

مثال

می‌خواهیم چهار ماتریس را برای درهم ضرب کنیم (اندازه ماتریس‌ها در پایین آن‌ها آمده است):

$$M = \frac{M_1}{[10 \times 20]} \times \frac{M_2}{[20 \times 50]} \times \frac{M_3}{[50 \times 1]} \times \frac{M_4}{[1 \times 100]}$$

برای حل بازگشته این مسئله، ایندا زیرمسئله را تعریف کنیم. ضرب ماتریس‌های $M_{ij} = M_i \times M_{i+1} \times \dots \times M_{j-1}$ را زیرمسئله P_{ij} و تعداد کمینهی ضرب اعداد حقیقی برای آنرا $c[i, j]$ می‌نامیم. اگر آخرین ضربی که در P_{ij} انجام می‌شود بین ماتریس M_k و M_{k+1} ($i \leq k < j$) باشد. در آن صورت مسئله به دو زیرمسئله P_{ik} و P_{kj} تبدیل می‌شود و $M_{ij} = M_{k+1, j} \times M_{ik} + c[i, j] + d_i d_k d_j$. و در آن صورت $c[i, j] = c[i, k] + c[k+1, j] + d_i d_k d_j$. در این فرمول d_i هزینه‌ی ضرب M_{ik} است.

برای پیدا کردن بهترین & نایاب زیرمسئله‌های فوق را به صورت بازگشتی برای همه‌ی i, j حل کنیم و حالت کمینه را پیدا کنیم. یعنی

$$c[i, j] = \begin{cases} \min_{i \leq k < j} \{c[i, k] + c[k+1, j] + d_i d_k d_j\} & \text{if } i < j \\ \infty & \text{if } i = j \end{cases} \quad (1)$$

این الگوریتم در واقع همه‌ی حالات ضرب n ماتریس را مررسی می‌کند و بین آن‌ها حالت بهینه را به دست می‌آورد. هزینه‌ی آن مناسب با تعداد حالات ممکن آن است که آنرا با $T(n) = \Omega(n^3)$ نمایش می‌دهیم. با توضیحات بالا داریم:

$$T(n) = \begin{cases} 1 & n = 1 \\ \sum_{i=1}^{n-1} T(i)T(n-i) & n > 1 \end{cases}$$

می‌توان نشان داد که $C(n) = C(n-1)^2$ است که برابر است با

$$C(n) = \frac{1}{n+1} \binom{n}{n} = \Omega(n^{\frac{3}{2}})$$

همچنین روش است که راه حل بازگشتی تعداد زیادی زیرمسئله‌ی تکراری را حل می‌کند. این همان ویرگی است که برای حل پریا می‌باشد.

راه حل بوا

این راه حل در واقع حل از پایین به بالای درخت بازگشت مربوط به حل بازگشته است، که در آن نتایج زیر مسئله‌های کوچک‌تر در ماتریس ذخیره می‌شوند تا برای محاسبه زیر مسئله‌های بزرگ‌تر مورد استفاده قرار گیرند.

ماتریس $C[1..n, 1..n]$ را تعریف می‌کیم که $C[i, j] = \min_{k=1}^{j-1} C[i, k] + C[k+1, j] + d[i] \cdot d[k] \cdot d[j]$. این داده‌های هر سه‌ی بهمه‌ی محاسبه ماتریس $R[1..n, 1..n]$ است. هم‌چنین $R[i, j] = \min_{k=1}^{j-1} R[i, k] + R[k+1, j] + d[i] \cdot d[k] \cdot d[j]$ است. مقدار k را نشان می‌دهد که حل بهمه‌ی مسئله مسخر به حرف دو زیر مسئله‌ی $M_{i,k} \times M_{k,j}$ برای $j < k \leq n$ شود. اساس الگوریتم بوا همان فرمول شماره‌ی ۱ است. بنابراین الگوریتم بوا به صورت زیر خواهد بود:

```

for i:=1 to n do C[i,i] := 0;
for l:= 2 to n do {number of matrices multiplied}
    for i:= 1 to n-l+1 do begin
        j:= i + l -1;
        {solving Mij}

        for k:= i to j-1 do begin
            C[i,j] := min {C[i,k] + C[k+1,j]} + d[i]*d[k]*d[j];
            R[i,j] := the value of k that makes the minimum
        end;
    end
end

```

با استفاده از ماتریس R می‌توانیم نحوه ضرب ماتریس‌ها را به صورت پرانتگذاری شده بددست آوریم. این کار با فراخوانی رویه‌ی Print_Results(i,n) انجام می‌شود:

```

Procedure Print_Results(i,j:integer);
begin
    if i=j then
        return
    else begin
        k:= R[i,j];
        write('(');
        Print_Results(i,k);
        write(' x ');
        Print_Results(k+1,j);
    end
end

```

بارویه‌ی مسابه می‌توان ماتریس حاصل را بددست آورد. روس است که زمان اجرای الگوریتم بوا از $O(n^3)$ و مقدار حافظه‌ی مصرفی آن از $O(n^2)$ است.

مثال

می خواهیم شش ماتریس با اندازه های زیر را در هم ضرب کنیم:

$$\frac{M_1}{[30 \times 25]} \times \frac{M_2}{[25 \times 15]} \times \frac{M_3}{[15 \times 5]} \times \frac{M_4}{[5 \times 10]} \times \frac{M_5}{[10 \times 20]} \times \frac{M_6}{[20 \times 25]}$$

در انتهای الگوریتم ماتریس های C و R در شکل ۴.۴ نشان داده شده است. در این شکل ماتریس ها دوران داده شده اند و فقط بخش بالا شش آنها نوشته شده اند. این شکل به خوبی نحوه محاسبه هزینه بینهای همه زیر ماتریس های R/M را نشان می دهد. مثلاً $C[2,5]$ باین صورت محاسبه می شود.

$$C[2,5] = \min \begin{cases} C[2,2] + C[2,2] + d_1 d_2 d_3 = 0 + 2500 + 25 \cdot 10 \cdot 20 & = 12000 \\ C[2,2] + C[4,0] + d_1 d_2 d_3 = 2625 + 1000 + 25 \cdot 5 \cdot 20 & = 7125 \\ C[2,2] + C[0,0] + d_1 d_2 d_3 = 1275 + 0 + 25 \cdot 10 \cdot 20 & = 11275 \\ & = 7125. \end{cases}$$

۴.۴.۴ مثلث بندی بینهای یک چندضلعی محدب

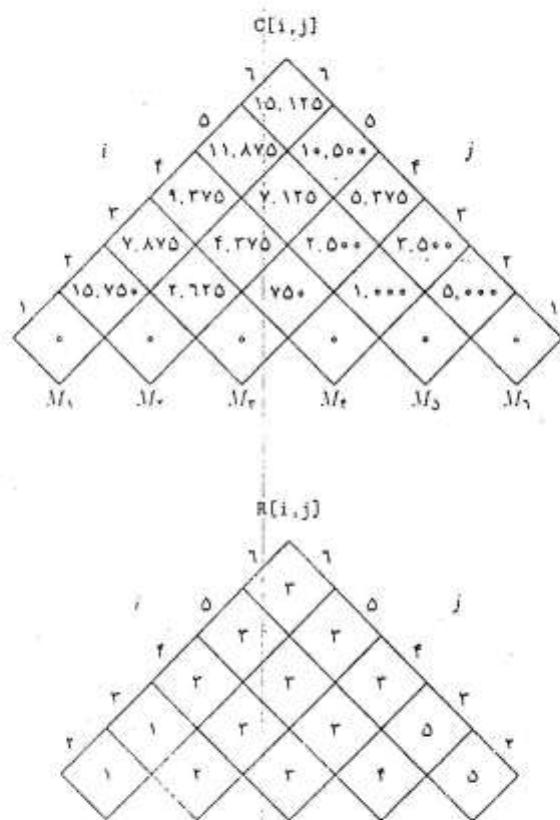
یک چندضلعی محدب را با دنباله $P = \langle v_0, v_1, \dots, v_{n-1}, v_n \rangle$ از رأس های آن در خلاف جهت عقربه های ساعت نشان میدهیم (شکل ??) که ضلع های آن $\langle v_0, v_1 \rangle, \langle v_1, v_2 \rangle, \dots, \langle v_{n-1}, v_n \rangle$ هستند. چون چندضلعی محدب است، اگر v_i و v_j مجاور هم نباشند، پاره خط $\langle v_i, v_j \rangle$ که به آن «قطر» می گوییم، حتماً در داخل چندضلعی قرار می گیرد. قطر $\langle v_i, v_j \rangle$ چندضلعی را به دو چندضلعی محدب $\langle v_0, v_1, \dots, v_i \rangle$ و $\langle v_j, v_{j+1}, \dots, v_{n-1}, v_n \rangle$ تقسیم می کند.

منظور از مثلث بندی یک چندضلعی تعیین مجموعه T از قطعه های نامقاطع است که چندضلعی را به مثلث های مختلف تقسیم کند. بک «ضلعی همواره با $\langle v_i, v_{i+1}, \dots, v_{i+k} \rangle$ قطب به i - مثلث افزایش می شود (جر؟)» در یک چندضلعی $\langle v_0, v_1, \dots, v_{n-1}, v_n \rangle$ می توان یک نایع وزن λ بر روی ضلع ها، قطعه ها یا مثلث های حاصلی تعریف کرد. مثلث بندی بی که در آن که حاصل جمع وزن ها کمتر شود را مثلث بندی بینه می گویند. طول مثلث های می تواند یک نایع وزن باشد نهونه دیگر وزن مثلث هاست که تقریبی هر مثلث به صورت ریز تعریف شود.

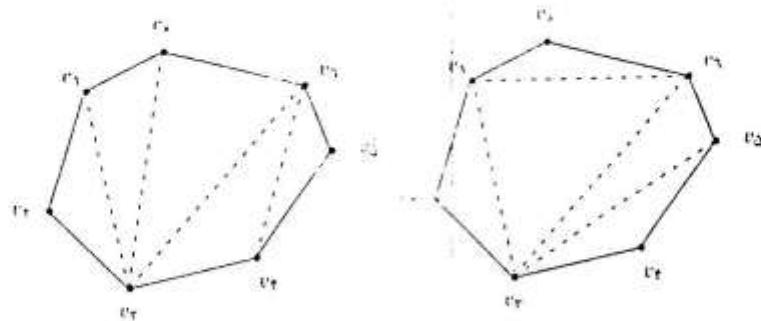
$$T = \langle v_0, v_1, \dots, v_{n-1}, v_n \rangle$$

آنچه ماضی اطلاعی بینهای است

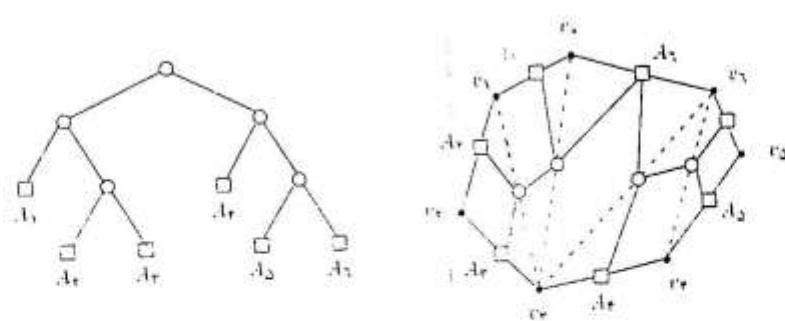
آنچه حداقلی اطلاعی بینهای است این است که از طریق یک درخت بینهای v_0, v_1, \dots, v_n بینهای قابل نایابش است. این درخت را می توان $T(v_0, v_1, \dots, v_n)$ نامید. ترتیب v_0, v_1, \dots, v_n را در درخت $T(v_0, v_1, \dots, v_n)$ می توان از v_0 شروع کرد. مثلاً شکل ۴.۴.۴.۲ ترتیب (v_0, v_1, \dots, v_n) را در ضرب $M_1 M_2 M_3 M_4 M_5 M_6$ می نماید. مکعبه (ماتریس) از هشت ایمیت است. اکثر شاخه های سمت چپ هر گره ترتیب ایمیت که در مذکور معادل ضرب $M_1 M_2 M_3 M_4 M_5 M_6$ است.

شکل ۴.۴: ماتریس‌های C و R برای متال ضرب ۶ ماتریس.

۴.۷ روش برنامه‌ریزی پریا



شکل ۴.۷: یک چندضلعی محدب و دو نوع متلبندی آن.



شکل ۴.۸: تاظر مسئله‌های خرا - مازرس - شو - متلبندی یک چندضلعی محدب.

همان‌طور که ملاحظه می‌شود ارتباط یک به بک مین درخت پارس و عبارت برآشگذاری شده وجود دارد. مثلفبندی یک چندضلعی بزیر با استفاده از درخت پارس قابل تماش است. شکل ۲۷.۴ ب معادل بودن این مسئله‌ها را نشان می‌دهد. گره‌های داخلی درخت، قطعه‌های چند ضلعی هستند و ضلع <۱۰, ۱۶> ب معادل بودن این محصور می‌شود. برگ‌ها و گره‌های خارجی ضلع‌ها هستند. زیره درخت یک ضلع مثلث <۵۰, ۱۲۱> در نظر گرفته شده است. این مثلث در واقع فرزندان ریشه را مشخص می‌کند که یکی از آن‌ها <۲۰, ۱۶> و دیگری <۴۰, ۱۶> می‌باشد. اگر دفت کنیم متوجه می‌شویم که مثلث مذکور چندضلعی اولیه را به سه قسم تقسیم کرده است: خود مثلث و دو چندضلعی که در راس α_2 متشکل‌کند. چند ضلعی‌های جدید که در بالا به آن‌ها اشاره شد با اصلاح اصلی چندضلعی اولیه به وجود آمده‌اند به جزءی از آن‌ها که همان قطعه‌ای <۱۰, ۱۶> و <۱۲, ۱۶> هستند. در حالت کلی چند ضلعی مثلث بندی شده با «ضلع با ۱ - برگ در درخت پارس ارتباط یک به بک دارد. پس حاصل ضرب زنجیری از n ماتریس متناظر با درخت پارس اند گره‌ای است و بنابراین متناظر با چندضلعی مثلثبندی شده با $n+1$ رأس است. ماتریس A در برنامه‌ی ضرب ماتریس‌ها متناظر با ضلع < $\alpha_0, \alpha_1, \alpha_2, \dots, \alpha_n$ > در چندضلعی $n+1$ رأسی است. قطر < $\alpha_0, \alpha_1, \dots, \alpha_n$ > (یعنی (i) متناظر است با ماتریس A_{n+1}) در هنگام محاسبه می‌باشد. در حقیقت مثلفبندی ضرب ماتریس‌ها حالت خاصی از مثلث بندی بهینه‌ی یک چندضلعی محدب است. یعنی هر عملیات که در ضرب ماتریس‌ها انجام می‌شود قابل اجرا در مثلث بندی است.

اگر زنجیری از ماتریس‌های A_1, A_2, \dots, A_n داشته باشیم چند ضلعی با $n+1$ رأس < $v_0, v_1, v_2, \dots, v_n$ > معادل آن است. اگر ماتریس A با ابعاد $d_1 \times d_2 \times \dots \times d_n$ باشد و زیر را برای مثلث بندی تعریف می‌کنیم:

$$w(\Delta(v_i, v_j, v_k)) = d_i d_j d_k$$

مثلثبندی بهینه چندضلعی P با درنظر گرفتن تابع فوق درخت پارس برای برآشگذاری بهینه‌ی $\alpha_1, \alpha_2, \dots, \alpha_n$ را ایجاد می‌کند.

هر چند حالت عکس صحت ندارد (مثلثبندی حالت خاص ضرب ماتریس‌ها نمی‌باشد) ولی پس از مررسی می‌بینیم که با کمی تغییرات در برنامه‌ی ضرب ماتریس‌ها می‌توانیم مثلث بندی را انجام دهیم. کافی است ابعاد ماتریس‌ها یعنی d_1, d_2, \dots, d_n را با رأس‌های $v_0, v_1, v_2, \dots, v_n$ جایه‌جا کنیم و تمام رجوع‌ها به i تبدیل به v_i شوند پس:

$$\min = m[i, k] + w[k+1, j] + w(\Delta(v_i, v_k, v_j))$$

و $m[1, n]$ ارزش مثلثبندی بهینه را دارد.

تعریف ساختار مثلث بندی بهینه (مرحله‌ی اول)

اگر مثلثبندی بهینه T با $n+1$ رأس چندضلعی < $\alpha_0, \alpha_1, \alpha_2, \dots, \alpha_n$ > در نظر گیریم که مثلث ارش مثلفبندی شده‌ی زیر چندضلعی‌های بوجود آمده خواهد بود. پس مثلثبندی زیر چندضلعی‌های بوجود آمده باید بهینه باشد زیرا اگر چندضلعی دیگری با ارزش کمتر وجود داشته باشد با بهینه بودن آن تناقض است.

۴.۴ روش برآمده بزرگی پرها

تعزیت بازگشتی (مرحله دوم)

همان طور که در ضرب ماتریس ها $[z_1, z_2, \dots, z_n]$ را هزینه های بهینه برای ضرب ماتریس های $[v_1, v_2, \dots, v_n]$ در نظر گرفته بس از i, j که $n \leq i \leq j \leq n$ را به عنوان ارزش مثبت بندی بهینه برای چند ضلعی $>[v_1, v_2, \dots, v_n]$ < منظور می کنیم. برای سادگی کار چند ضلعی (یک جمله) $>z_1, z_2, \dots, z_n<$ را با ارزش صفر تعريف می کنیم و ارزش مثبت بندی بهینه P را با $[1/n]$ مشخص می کنیم. پس $=0$ $>z_1, z_2, \dots, z_n<$ و زمانی که $1 \leq i \leq j \leq n$ - زیک چند ضلعی با حداقل ۲ راس داریم که برای همه راس های v_k که $i \leq k \leq j$ و v_k را با ارجاعیم ارزش مثبت $\Delta(v_{i-1}, v_j, v_k)$ دارد به اضافه هی چند ضلعی های بوجود آمده $>v_i, v_{i+1}, \dots, v_j<$ و $>v_{i-1}, v_i, \dots, v_j<$ را به حداقل بررسیم. فرمول بازگشتی عبارت است از:

$$t[i, j] = \begin{cases} 0, & i = j \\ \min_{1 \leq k \leq j-i} \{ t[i, k] + t[k+1, j] + w(\Delta(v_{i-1}, v_j, v_k)) \}, & i < j \end{cases}$$

همانند الگوریتم ضرب ماتریس ها این عملیات بازمان $O(n^3)$ و فضای $O(n^2)$ انجام می شود و همان کد *MatrixChainOrder* با اندکی تغییرات قابل استفاده برای مثبت بندی می باشد.

۴.۴.۴ بزرگترین زیردنباله می مشترک

زیردنباله یک دنباله داده شده، دنباله ای است که با حذف تعدادی (شاید هیچ) از عناصر آن دنباله ایجاد می شود. به عبارتی دیگر، $Z = <z_1, z_2, \dots, z_m>$ زیردنباله $X = <x_1, x_2, \dots, x_n>$ است اگر دنباله ای اکیداً صعودی $>x_1, x_2, \dots, x_n<$ از اندیس های عناصر X وجود داشته باشد به طوری که برای $k = 1, \dots, n$ x_k ز داشته باشیم: $x_k = z_i$. مثلاً $Z = < B, C, D, B >$ بزرگترین دنباله از $X = < A, B, C, B, D, A, B >$ است که دنباله ای اندیس های مریوط $< 2, 3, 5, 7 >$ می باشد.

مسئله مورد نظر در این بخش پیدا کردن Z «بزرگترین زیردنباله می مشترک» (Longest Common Subsequence (LCS)) برای دو دنباله داده شده X و Y است. منظور از بزرگترین طولانی ترین دنباله است. مثلاً اگر $X = < A, B, C, B, D, A, B >$ و $Y = < B, D, C, A, B, A >$ باشد $< B, C, A >$ به طول ۳ بزرگترین می مشترک برای X و Y است ولی بزرگترین آنها نیست. بزرگترین زیردنباله می مشترک این دو دنباله به طول ۴ است و $< B, C, B, A >$ یک جواب آن است.

به طور دقیق تر، ورودی دو دنباله $X = < x_1, x_2, \dots, x_m >$ و $Y = < y_1, y_2, \dots, y_n >$ و هدف پیدا کردن LCS این در دنباله است. اگر هر دنباله یک فایل متی و هر عنصر آن یک سطر از آن فایل باشد، LCS این دو فایل اکتبا می باشد این است که دستور `diff file1 file2` در سیستم عامل بوئیکس در خروجی می نویسد.

در اینجا راه حل این مسئله بررسی کلیه زیردنباله های X و Y و مقایسه آن هاست که به وضوح الگوریتمی طبقه بندی می دهیم که این مسئله دارای زیرمسئله های بهینه است و راه حل بازگشتی آن منجر به حل مسئله اصلی می شود. این دو ویژگی ما را به راه حل پویا برای این مسئله را همایی می کند.

معرف

اگر $i = 1..m$ برای $X = < x_1, x_2, \dots, x_m >$ یک دنباله باشد، آنین پیشوند (prefix) برای $X_i \equiv < x_1, x_2, \dots, x_m >$ نعرف می‌کنیم. مثلاً برای $X = < A, B, C, B, D, A, B >$ داریم:

$$X_4 = < A < B < C < D >$$

قضیه ۱۱. اگر $X = < x_1, x_2, \dots, x_m >$ و $Y = < y_1, y_2, \dots, y_n >$ دنباله‌های ورودی باشند و $LCS(X, Y)$ باشد داریم:

$$\text{اگر } x_m = y_n, \text{ داریم } Z_{k-1} \cup z_k = x_m = y_n \text{ است.}$$

اگر $x_m \neq y_n$ ، آن‌گاه از $x_m \neq y_n$ نتیجه می‌گیریم که Z برای $X_{m-1}, Y_{n-1}, LCS(X_{m-1}, Y_{n-1})$ است.

اگر $y_n \neq x_m$ ، آن‌گاه از $y_n \neq x_m$ نتیجه می‌گیریم که Z برای $X_{m-1}, Y_{n-1}, LCS(X_{m-1}, Y_{n-1})$ است.

اثبات.

۱. اگر این چنین نباشد، یعنی $x_m \neq z_k$ ، می‌توانیم با اضافه کردن $y_n = x_m$ به عنوان آخرین عنصر Z ، یک زیردنباله‌ی مشترک بزرگ‌تر از قبل ایجاد کنیم که با این فرض که Z برای $X_{m-1}, Y_{n-1}, LCS(X_{m-1}, Y_{n-1})$ است تناقض دارد.

۲. اگر $x_m \neq z_k$ ، پس Z برای $X_{m-1}, Y_{n-1}, LCS(X_{m-1}, Y_{n-1})$ است. اگر Z برای $X_{m-1}, Y_{n-1}, LCS(X_{m-1}, Y_{n-1})$ نباشد؛ نمی‌تواند طولی از k بیشتر باشد، چرا که با فرض Z بودن Z تناقض دارد.

۳. مشابهی بند (۲).

□

راه حل بازگشتی

از قضیه ۱۱ می‌توان دید که راه حل بازگشتی مسئله سنجربه یک با دو فراخوانی بازگشتی خواهد شد. اگر $x_m = y_n$ ، باید $LCS(X_{m-1}, Y_{n-1})$ را برای X_{m-1}, Y_{n-1} پیدا کرد و به آن x_m اضافه کرد. اگر $x_m \neq y_n$ ، دو فراخوانی بازگشتی برای حل زیرمسئله‌های به دست آوردن $LCS(X_{m-1}, Y_{n-1})$ و $LCS(X_{m-1}, Y_{n-1})$ و X_{m-1}, Y_{n-1}, X انجام داد. بزرگ‌ترین زیردنباله‌ای این دو زیرمسئله LCS پاسخ است.

روشن است که راه حل بازگشتی مسخر به حل تکراری زیرمسئله‌های مختلف می‌شود. بنابراین راه حل پویا برای این مسئله مناسب است. اگر $[i..j]$ طول LCS برای $X_i..X_j$ و $Y_i..Y_j$ باشد، فرمول ۲ ارضاً این مقادیر را برای زیرمسئله‌های مختلف نشان می‌دهد.

$$r[i, j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ r[i - 1, j - 1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j, \\ \max(r[i - 1, j], r[i, j - 1]) & \text{if } i, j > 0 \text{ and } x_i \neq y_j. \end{cases} \quad (2)$$

راه حل پویا

براساس فرمول ۲ می‌توان الگوریتم پویای زیر را برای این مسئله ارائه داد. برای این کار نیاز به ماتریس $c[0..m, 0..n]$ داریم که $c[j][i]$ طول LCS برای $X_i..X_j$ و $Y_i..Y_j$ در خود ذخیره می‌کند. مانند پیش راه حل های پویا

۴.۴ روش برنامه‌ریزی پویا

برای ساختن LCS ماتریس $b[i..m, j..n]$ را تعریف می‌کنیم که $b[i..j]$ به صورت نمایین حاوی برداری است که نشان می‌دهد که $[i..j]$ از کدامیک از زیرمسئله‌ها ساخته شده است. اگر $b[i..j] = \emptyset$ در آن صورت $c[i..j]$ از $c[i..j-1]$ ساخته شده است. اگر $c[i..j] = c[i..j-1] \cup c[i..j]$ و $b[i..j] = \leftarrow$ اگر $c[i..j] = c[i..j-1] \cup c[i..j-1]$ بودست آنده است.

```

for i := 1 to m do c[i, 0] := 0;
for j := 1 to n do c[0, j] := 0;
for i := 1 to m do
    for j := 1 to n do
        if  $x_i = y_j$  then begin
            c[i, j] := c[i - 1, j - 1] + 1;
            b[i, j] := '^';
        end
        else if c[i - 1, j] ≥ c[i, j - 1]
            then begin
                c[i, j] := c[i - 1, j];
                b[i, j] := '↑';
            end
        else begin
            c[i, j] := c[i, j - 1];
            b[i, j] := '←';
        end
    end
end

```

مثال

اگر $Y = < B, D, C, A, B, A >$ و $X = < A, B, C, B, D, A, B >$ باشد مطابق شکل ۳۸.۴ خواهد بود.

روشن است که الگوریتم ارائه شده از مرتبه زمانی $O(mn)$ و میزان حافظه مخفی آن نیز $O(mn)$ است.

LCS ساختن

با توجه به اطلاعات ذخیره شده در ماتریس b می‌توان Print-LCS(b, X, m, n) را با فراخوانی $LCS(b, X, m, n)$ ساخت. این روش به صورت زیر است:

$i \setminus j$	0	1	2	3	4	5	6
0	*	*	*	*	*	*	*
1	A	*	*	*	*	*	*
2	B	*	*	*	*	*	*
3	C	*	*	*	*	*	*
4	D	*	*	*	*	*	*
5	A	*	*	*	*	*	*
6	B	*	*	*	*	*	*

شکل ۳.۴ ماتریس‌های c و b در پیداکردن LCS برای $X = < A, B, C, B, D, A, B >$ و $Y = < B, D, C, A, B, A >$

```

PRINT-LCS( $b, X, i, j$ )
if ( $i = 0$  or  $j = 0$ )
    then return;
if  $b[i, j] = ^\nwarrow$ 
    then begin
        PRINT-LCS ( $b, X, i - 1, j - 1$ );
        print  $x_i$ 
    end
else if  $b[i, j] = ^\uparrow$ 
    then PRINT-LCS ( $b, X, i - 1, j$ )
    else PRINT-LCS ( $b, X, i, j - 1$ )

```

این الگوریتم از مرتبه‌ی $O(m + n)$ است.

۵.۴.۴ درخت دودویی جستجوی بهینه (Optimal BST)

درخت جستجو (Binary Search Tree) یک ساختمان داده‌ی مناسب برای پیاده‌سازی فرهنگ‌های داده‌ای است. برای فرهنگی با n عنصر، اعمال درج، حذف و جستجو را می‌توان با متوسط $O(\log n)$ انجام داد.

۴.۳ روش برنامه ریزی پروژا

اگر داده‌ها از پیش آمده باشند، می‌توانیم آن‌ها را به گونه‌ای در درخت قرار دهیم، تا درخت متوازن بودست آید.
 بدین ترتیب می‌توان اعمال درج، حذف و جست‌وجو را در بدترین حالت با $O(\log n)$ انجام داد. برای ساختن چنین درختی، ناید عنصر میانه را در ریشه قرار دهیم. بدین ترتیب تبعی از عناصر در زیر درخت چپ و یعنی دیگر در زیر درخت راست فراز می‌گیرند. سپس همین روش را روی زیر درخت‌های چپ و راست انجام می‌دهیم.
 بدین ترتیب اتفاق می‌افتد. مثلاً اگر اطلاعات مری بوت به یک کتابخانه را درخت دودویی جست‌وجویی پیاده‌سازی کنیم، کار اتفاق می‌افتد. برروی آن عمل جست‌وجوی است و این جست‌وجوها می‌توانند «موفق» یا «ناموفق» باشند! یعنی کتاب در کتابخانه موجود باشد یا خیر. نکته‌ی مهم در اینجا آن است که در عمل تعداد تقاضاهای جست‌وجوی عناصر مختلف این درخت (کتاب‌های مختلف) پیاده‌سازی شفاقت است. با توجه به آن که زمان انجام جست‌وجوی برای هر یک عنصر مناسب با عمق آن عنصر در درخت است، اگر بتوانیم عناصری را که نعداد جست‌وجوی را دورتر از ریشه قرار دهیم، متوسط زمان به ریشه نزدیک‌تر کنیم و آن‌ها را که کمتر مورد جست‌وجو قرار می‌گیرند را دورتر از ریشه قرار دهیم، متوسط زمان جست‌وجو کمتر از حنا حالت متوازن خواهد شد. این مطلب، ایده‌ی اصلی درخت دودویی جست‌وجوی بهجه است.
 در شکل ?? یک درخت دودویی جست‌وجوی متوازن برای عناصر $7 < 4 < 1 < 5$ دیده می‌شود. فرض کرد $P(n)$ احتمال آن است که یک جست‌وجوی دلخواه برای عناصر «باشد. و فرض کرد که

$$\begin{array}{lll} P(a_3) = \frac{1}{\tau\gamma}, & P(a_T) = \frac{\tau}{\tau\gamma}, & P(a_F) = \frac{\tau}{\tau\gamma} \\ P(a_T) = \frac{\tau}{\tau\gamma}, & P(a_3) = \frac{1}{\tau\gamma}, & \\ P(a_F) = \frac{\tau}{\tau\gamma}, & P(a_T) = \frac{\tau}{\tau\gamma}, & \end{array}$$

با نتیجه به آن که زمان چشت و جو سرای هر عنصر مناسب است با عنوان آن عنصر در درخت، متوجه شد که درخت بر این است:

$$\frac{1}{27}[(1+1)+(2+2)+2+(1+2+3+4)+2] = \frac{62}{27}$$

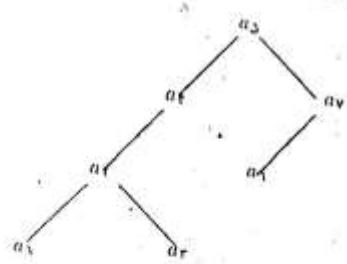
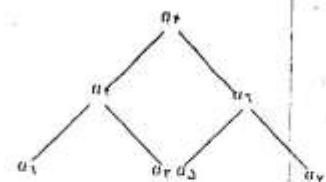
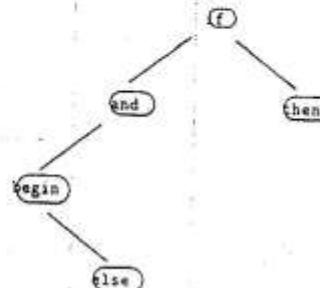
اگر به جای این درخت، درخت شکل ؟؟ را برای این عناصر ابجاد کیم، ما آن که ارتفاع درخت بیش نر شده است، متوسط زمان جست وجو کمتر می شود:

$$\frac{1}{T-1} [7 + 1 + (T+V) + T + (T-V) + 2 + (1+V) + F] = \frac{58}{T-1}$$

به عنوان مثالی دیگر، توجه کنید که هر کلمه‌ای استفاده شده در متون یک زبان برنامه‌نویسی با کلمات موجود در جدول نماد (symbol table) آن کامپایلر جست و جو می‌شود و این کار به دفعات زیادی انجام می‌شود. مثلاً در یک زبان پاسکال، ممکن است یک جدول نماد از ۵ کلمه‌ای کلیدی به صورت شکل ۴-۰ ساخته شود.

جست و جوهای ناموفق برای پیدا کردن عناصری که در درخت نیستند هم در شکل بهینه‌ی درخت نافری می‌گذاردند و باید آن ها را در نظر گرفت. این جست و جوهای را در پکا درخت دودویی جست و جو می‌توان با عناصر خارجی آن نشان داد. مهارای هر اشاره‌گر Nil در درخت یک عنصر خارجی قرار می‌دهیم. مثلاً در شکل ۴-۰، اگر عناصر خارجی را `!`` بنامیم، با نشان دهندهٔ جست و جو برای کلیدی کلمات کرچکتر از `begin` است، با نشان دهندهٔ

جست و جو برای کلیدی کلامی است که از `begin` بزرگتر و از `else` کمتر است.

شکل ۴.۴: درخت نامتعارن با متوسط زمان جستجوی $\frac{n}{2}$.شکل ۴.۵: درخت متوازن با متوسط زمان جستجوی $\frac{n}{2}$.

شکل ۴.۶: یک درخت دودویی جستجو برای کلمات کلیدی.

مساله: ثابت کند تعداد عناصر خارجی برای یک درخت دودویی جستجو با n عنصر، $1 + n$ است.

برهان:

۱. استفاده از روش استقرانی

۲. بدانای هر عنصر غیر از ریشه، یک پدر و در نتیجه یک بال تناظر وجود دارد. (در کل $1 - n$ بال) از طرفی

دیگر هر عنصر دو فرزند دارد که برخلاف از فرزند های عناصر خارجی هستند. بنابراین $2n$ بال به عناصر داخلی و خارجی وارد می شوند. پس $1 - n = n + 1 = n + 1$ عنصر به عنوان عنصر خارجی باقی می ماند.

۳. اگر n عنصر درخت داشته باشد، آنگاه $1 + n$ باره بین آنها برای عناصر خارجی موجود است:

$$b_1 < a_1 < b_2 < a_2 < \dots < a_n < b_n$$

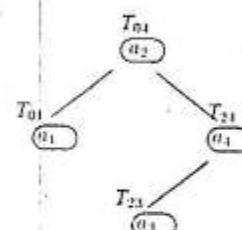
ساختن درخت دودویی جستجوی بهینه

فرض کرد که

۴.۴ روش برنامه‌ریزی پوشا

					c_{ij}
*	$c_{00} = 0$ $w_{00} = T$	$r_{01} = 1$ $w_{01} = 1$ $r_{01} = a_1$	$c_{0T} = 1A$ $w_{0T} = 1T$ $r_{0T} = a_1$	$r_{0F} = 10$ $w_{0F} = 1F$ $r_{0F} = a_1$	$c_{0F} = 1Y$ $w_{0F} = 1Y$ $r_{0F} = a_T$
V		$r_{11} = 0$ $w_{11} = T$	$c_{1T} = 1$ $w_{1T} = 1$ $r_{1T} = a_T$	$c_{1F} = 11$ $w_{1F} = A$ $r_{1F} = a_T$	$c_{1F} = 1A$ $w_{1F} = 10$ $r_{1F} = a_T$
T			$c_{TT} = 0$ $w_{TT} = 1$	$c_{TF} = T$ $w_{TF} = T$ $r_{TF} = a_T$	$c_{TF} = A$ $w_{TF} = 0$ $r_{TF} = a_T$
T				$c_{TF} = 0$ $w_{TF} = 1$	$c_{TF} = T$ $w_{TF} = T$ $r_{TF} = a_T$
F					$c_{TF} = 0$ $w_{TF} = 1$

جدول ۴.۴: مقادیر r_{ij}, w_{ij}, c_{ij} برای مثال.



شکل ۴.۴: درخت دودویی جستجوی بهینه برای مثال.

۶.۴.۴ مسئله کوله پشتی (Knapsack Problem)

کوله‌پشتی از مسئله‌های کلاسیک و مهم و با کاربرد زیاد است که می‌تواند به این صورت مطرح شود: دردی به منظور سرفت به فروشگاهی می‌رود. در فروشگاه تعدادی بسته از جنس‌های مختلف وجود دارد. هر جنس ارزش مشخصی دارد و وزن آن نیز معلوم است. در برای برداشتن اجنس یک کوله پشتی به همراه دارد که با آن می‌تواند تا حد اکثر ۱۷ کیلوگرم را حمل کند (وزن‌ها فرمگی اعداد صحیح هستند). به فرض این که درد نمی‌تواند قسمی از یک بسته را بردارد. الگوریتمی طراحی کنید که مشخص کند که درد باید کدامیک از بسته‌ها را بردارد که مجموع وزن آن‌ها از ۱۷ کیلوگرم بیشتر نشود و مجموع ارزش آن‌ها نیز بیشترین مقدار ممکن شود. با توجه به این که در این مسئله‌ی برداشتن جزئی از یک بسته محابز نیست و درد تنها می‌تواند یک بسته را به تمامی بردارد و با اصل‌آن را برندارد، به این مسئله، مسئله کوله پشتی سفروپک (0-1 Knapsack) نیز می‌گویند. می‌خواهیم بینیم که آیا برای این مسئله راه حل سریع وجود دارد.

مسئله را می‌توان در حالت کلی به صورت زیر تعریف کرد:

ورودی:

- یک کوله‌پشتی با توانایی حمل M . واحد وزن جنس

- N نوع جنس

- تعداد ستدی از بار نوع i ام باره N_i (یا $\text{Number}[i]$ در برنامه‌ها)

- وزن جنس نوع i ام w_i (یا $\text{weight}[i]$ در برنامه‌ها). فرض می‌کیم که $w_i \leq M$

- ارزش جنس نوع i ام C_i (یا $\text{Cost}[i]$ در برنامه‌ها)

فرض: w_i, C_i اعداد صحیح هستند.

می‌خواهیم کوله‌پشتی را با این بارها کامل‌با ناحد امکان پر کیم به طوری که اگر بارها ارزش داشته باشند و مجموع ارزش بارها بیشینه شود.

حالت اول:

$C_i = 0$ و کوله‌پشتی برشود.

این مسئله یک مسئله NP-نمای است و راهی به جز بررسی تمام حالت‌ها ندارد ولی باید با نظم خاصی این حالت‌ها را بررسی کیم. برای این منظور ایندا از روش پس گرد (Backtracking) استفاده می‌کیم.

راه حل پس گرد

بارشماری i را در نظر بگیرید. این بار دو حالت ممکن است داشته باشد. یا جزء انتخاب‌هاست که در آن صورت وزن آن را از توانایی حمل کوله‌پشتی کم می‌کیم. یا جزء انتخاب‌ها نیست که در آن صورت بارهای بعدی را امتحان می‌کیم.

۳.۴ روش برنامه‌بازی پرها

برای حل این مسئله تابع Knapsack را برای پرگردان یک کوله‌پشتی به توانایی حمل M از جنس‌های شماره‌ی N می‌نویسیم. این تابع در صورتی که مسئله جواب داشته باشد True و در غیر این صورت False برمی‌گرداند.

```
function Knapsack(W, i: Integer): Boolean;
begin
  if W = 0 then return True;
  if (W < 0) or (i > N) then return False;
  { load #i is a candidate.}
  if Knapsack(W - Weight[i], i + 1) then
    begin
      WriteLn(i, Weight[i]);
      return True;
    end
  else return (Knapsack(W, i + 1));
end;
```

بدترین حالت هنگامی است که مسئله جواب نداشته باشد. در آن صورت الگوریتم از مرتبه‌ی $\Theta(2^N)$ است.

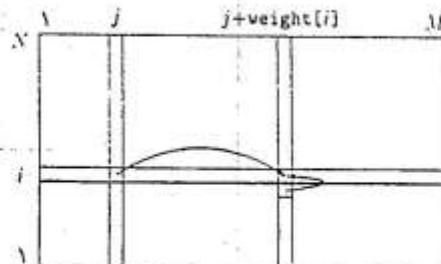
$$T(n) = 2T(n-1) + \Theta(1) \Rightarrow T(n) = \Theta(2^n)$$

راه حل کورکورانه

راه حل کورکورانه برای مسئله‌ی فوق بررسی تمام حالات ممکن به ترتیب زیر است: چنان‌چه بخواهیم یکی از وزنه‌ها را انتخاب کیم به جای آن ۱ و در غیر این صورت به جای آن صفر در نظر می‌گیریم. بنابراین 2^N حالت مختلف خواهیم داشت که هر حالت تبایش دودویی اعداد صفر تا $1 - 2^N$ می‌باشد. بنابراین می‌توانیم با یک حلقة تمام حالت‌های ممکن را آزمایش کیم:

```
for i := 0 to  $2^N - 1$  do
begin
  find the N-bit binary representation of i;
  find the loads with 1's in binary form of i;
  if the sum of weights of the selected loads
    is equal to M then this is one solution
end
```

در بدترین حالت این دو الگوریتم مانند هم عمل می‌کنند. و ما نوجه به NP-نمای بودن مسئله این امر قابل انتظار است.

شکل ۴۲.۴: نحوه پر کردن ماتریس S در الگوریتم پویا.

راه حل پویا

این مسئله به صورج دارای خصوصیت زیرساختار پهنه است: هرگاه به اندازه i درون گوله‌پشتی جنس فرار داده باشیم، پس از آین مسئله تبدیل می‌شود به پر کردن یک گوله‌پشتی با طرفیت $m - M$ با بسته‌هایی که باقی مانده‌اند. واضح است که هر جواب پهنه برای مسئله اصلی، شامل یک جواب پهنه برای آین مسئله نیز هست.

ماتریس $S[M \times N]$ را به صورت زیر تعریف می‌کنیم: $[j, i] = S[i, j]$ نشان می‌دهد که آیا می‌توان گوله‌پشتی با اندازه i ترا با بارهای از j تا M پر کرد یا خیر. اگر نتوان، این مقدار صفر است و در غیر این صورت برابر > 0 است که k نمایه‌ی آخرین باری است که در گوله‌پشتی فرار می‌دهیم.

در حالت کلی داریم:

$$S[i, j] = S[i + 1, j - weight[i]]$$

شکل ۴۲.۴ نحوه پر کردن ماتریس را نشان می‌دهد. این الگوریتم به شرح زیر است: (فرض $k = j + weight[i]$)

* در سطر اول داریم: $S[1, weight[1]] = 1$

* سطر i بر اساس سطر $i - 1$ پر می‌شود.

* ابتدا مقادیر سطر $i - 1$ در سطر i کپی می‌شود.

* $S[i, k]$ را از $[j, i, S[i, k]]$ پر می‌کنیم در صورتی که $S[i, k]$ نوبط بارهای قبلی پر نشده باشد.

* اگر $i = j$ یا $i < j$: $S[i, k]$ را فقط با یک عدد بار پر می‌کنیم.

روش برنامه ریزی پوچ

۱۷۵

```
for i := 1 to N do
    for j := 1 to M do S[i,j] := 0;
for i := 1 to N do
    for j := 0 to M - Weight[i] do
        begin
            if i > 1 then S[i, j] := S[i-1, j];
            k:=j + weight[i];
            if (j=0 or (S[i,j] > 0))
            then {we may be able to fill S[i,k]}
            if (k <= M) and (S[i,k] =0)
            then S[i,k] := i;
        end;
```

با پک آرایه همین کار را می توان انجام داد.

```
for j:= 1 to M do S[j] := 0;
for i := 1 to N do
    for j := 0 to M - Weight[i] do
        begin
            k:=j + weight[i];
            if (j=0 or ((S[j] > 0) and (S[j] <> i)))
            %load i can be used only once!
            then {we may be able to fill S[k]}
            if (k <= M) and (S[k] =0)
            then S[k] := i;
        end;
```

و پک جواب را هم می توان نوشت:

```
procedure print_results (j:integer);
{print the one solution for knapsack of size j
 we assume that this knapsack can be filled}
begin
    if j = 0 then return;
(*) k := S[j]; (k has to be > 0)
    writeln (k);
    print_results (j-k)
```

۱۷۶

به دلیل این که اگر مسئله $S[i, j]$ بیش از یک جواب داشته باشد (بکی با استفاده از بار i و دیگری با بارهای کمتر از آن) ما اولویت را به جواب $S[i-1, j]$ می‌دهیم، راه حل فوق درست است. یعنی اگر $i = [j\text{ا}]$ حتماً $j < k$ باشد.

حالت دوم (کلی):

$C_i = \infty$ ، $i < n$ ، و کوله‌پشتی کاملاً پرسود، در این حالت درایه‌ی $[j\text{ا}]S$ شامل دو مولفه است:

- شماره‌ی آخرین باری که در کوله‌پشتی S فزار می‌گرد، اگر این کوله‌پشتی را نتوان پرسود، این مولفه مبارابر Last است.
- عدد استفاده شده از بار $\text{last}[j\text{ا}]$ در این کوله‌پشتی.

برای i : $\text{Number}[i] = \text{تعداد بار از نوع } i$

شرح الگوریتم:

برای هر سطر i ام ماتریس را پرمی کنیم. در این صورت

- برای $i = j$ از $[j\text{ا}]S$ تحت شرایطی $S[j + weight[i]] = S[j]$ را با فزار دادن یک عدد دیگر از بار $M - weight[i]$ پرمی کنیم.

برای $i > j$ نقط در صورتی که قلاً پرسود، باشد، پرمی شود

برای $i < j$ نقط در صورتی که قلاً پرسود، باشد، پرمی شود

```

for j := 1 to M do S[j].last := 0;
for i := 1 to N do
  for j := 0 to M - weight[i] do
    begin
      k := j + weight[i]; {trying to fill S[k]}
      {conditions to use load i for the first time}
      if (j=0) or ((S[j].last > 0) and (S[j].last < i))
      then
        if (k <= M) and (S[k].last = 0)
        then begin
          S[k].last := i;
          S[k].No := 1
        end
        else
          {conditions to use more than one load i}
          if (S[j].last = i) and (S[j].No < Number[i])
          then
            if (k <= M) and (S[k].last = 0)
            then begin
              S[k].last := i;
              Inc (S[k].No)
            end
          end;
    end;
  end;
end;

```

رویه‌ی قبای با تغییر دستور (*) به print_results در این حالت درست کار می‌کند.

حالت سوم:

$n_i = \infty$, $C_i = 0$, و کل هشتگی کاملاً پر شود.
در این حالت بار نوع i می‌تواند بیش از یک بار استفاده شود.

```

S[0] := 0;
for i := 1 to N do
  for j := 0 to M do
    begin
      k := j + weight[i];
      if (j = 0 or S[j] < 0) then
        if k <= M then S[k] := i;
    end;
  end;

```

رویه‌ی print_results در این حالت نیز درست کار می‌کند.

مسئله‌ی خرد کردن ہول (اگر سخواهیم تعداد سکه‌ها کمیه باشد) یک مسئله‌ی کوله‌پشتی با $n_i = \infty$ است.

حالت چهارم:

$n_i > 0$ و $C_i > 0$ باشد و کوله‌پشتی کاملاً پر شود.

در این حالت یک مولفه‌ی جدید برای درایه‌ی $S[i,j]$ در نظر می‌گیریم:

$S[i,j].value$: بیشترین ارزش بارهای از نوع i با برای پر کردن کوله‌پشتی به اندازه‌ی j .

```

for i := 1 to N do
  for j:= 1 to M do S[i,j].value := 0;
for i := 1 to N do
  for j := 0 to M - Weight[i] do
    begin
      if i > 1 then S[i, j] := S[i-1, j];
      k := j + weight[i];
      if (j = 0 or (S[i-1, j].value > 0))
      then {we may be able to fill S[i,k]}
      if (k<=M) and (S[i,k].value < S[i-1,j].value + Cost[i])
      then begin
        S[i,k].last := i;
        S[i,k].value := S[i-1,j].value + Cost [i]
      end
    end;
  end;
```

جواب را بزر می‌توان نوشت:

```

procedure print_results (i, j: integer);
begin
  if j = 0 then return;
  k := S[i,j].last; {k has to be > 0}
  writeln (k);
  print_results (k-1, j-k)
```

مسئله را می‌توان با یک آرایه هم به صورت زیر حل کرد. ولی جنس‌های استفاده شده را نمی‌توان به دست آورد.
در این صورت، آرایه‌ی $M \times M$ همان ارزش است.

```

for j := 0 to M do S[j] := 0;
for i := 1 to N do
  for j := M downto weight[i] do
    begin
      if i = 1 then S[weight[i]] := Cost[i];
      else begin
        k := j - weight[i];
        if S[j] < S[k] + Cost[i] then
          S[j] := S[k] + Cost[i]
      end
    end;
  
```

برای پیدا کردن با ارزش ترین جواب ممکن است کوچکترین کاملاً بر نشود. در این حالت در مسیر N ام دستال
بزرگترین ارزش می‌گردد.

۵.۴ روش حریصانه در طراحی الگوریتم‌ها

روش حریصانه (Greedy) یکی از روش‌های ساده، جالب و سریع برای حل دقیق مسئله‌های است. البته باید مسئله دارای شرایط خاصی باشد تا نتوان از این روش برای حل آن استفاده کرد. مناسب‌ترین تعداد کمی از مسئله‌ها راه حل حریصانه را دارند. از جمله‌ای آن‌ها مسئله‌های مهمی هستند که راه حل تعدادی از آن‌ها به دلیل نوآوری قابل توجه بعنوان طراحی‌های آن‌ها نیت شده‌اند: مسئله الگوریتم هافمن (Huffman) برای پیدا کردن کدهای بهینه برای فشرده‌سازی متن‌ها، الگوریتم دایکسترا (Dijkstra) برای پیدا کردن همه‌ی کوتاه‌ترین مسیرها از یک رأس در یک گراف، و الگوریتم‌های برمیم (Prim) و کروکسکال (Kruskal) برای پیدا کردن درخت پوشای کمینه در یک گراف، و الگوریتم Fleury برای پیدا کردن یک دور اولی در گراف.

روش حریصانه اگر منجر به پیدا کردن جواب دقیق مسئله نشود یک روش نفربینی برای حل آن مسئله است که در صورتی که نتوان اثبات کرد که جواب بدست آمد، در بدترین حالت چه مقدار با جواب دقیق فاصله دارد، بسیار با ارزش است و یکی از روش‌های مهم در طراحی الگوریتم‌های نفربینی «قابل اثبات» است. در صورتی که چنین اثبات وجود نداشته باشد، روش حریصانه در واقع یک روش مکانیزمی (heuristic) برای حل مسئله است که ممکن است در عمل کاربر فراوان داشته باشد ولی از نظر علمی چنان‌جا ارزش نیست.

مسئله‌ای که با روش حریصانه قابل حل هستند دارای خصوصیات زیر می‌باشند:

- * مسئله بهینه‌سازی (optimization) است.
- * برای حل بهینه‌ی مسئله باید زیرمسئله‌های آن را بین‌بهینه‌سازی کرد (optimal subproblem).
- * مسئله‌ای که به روش بین‌بهینه حل بودند بین‌دو و بین‌گی نویی را داشتند.
- * انتخاب حریصانه (greedy choice) در این گونه مسائل بین‌بهینه انتخاب است و عوض نمی‌شود. (تفاوت با روش بین‌دو و بین‌گی است.)

به طور کلی می‌توان یک الگوریتم حریصانه را مطابق زیر بیان کرد:

```

function greedy (C: set): set
(C: the set of all candidates)
begin
  Makennull(S); (S is the solution)
  while not solution(S) and not Empty(C) do
    begin
      x := an element in C maximizing select(x);
      C := C - {x}
      if feasible (S U {x}) then S := S U {x}
    end
    if solution(S) then return (S)
    else return ("NO solution")
  end.

```

۵. روش حریصانه در طراحی الگوریتمها

۱.۵.۴ ویژگی‌های انتخاب حریصانه

انتخاب نامزد در صورت نداشتن نضاد با انتخاب‌های الجاوه شده انتخاب نهایی است و عمل پس گرد (backtracking) برای تعیین انتخاب صورت نمی‌گیرد. این انتخاب براساس تابع انتخاب (selection function) و براساس مقادیر محلی صورت می‌گیرد. در واقع الگوریتم حریصانه به گونه‌ای است که انتخاب مبتنی بر بهینه‌سازی محلی (local optimization) منجر به بهینه‌سازی سراسری (global optimization) می‌شود که هدف مسئله است.

۲.۵.۴ انتخاب فعالیت‌ها (Activity Selection Problems)

۱. فعالیت مختلف، a_1, a_2, \dots, a_n ، داده شده‌اند که همگی از یک منبع غیرقابل اشتراک استفاده می‌کنند. برای هر فعالیت دو پارامتر زمان شروع و زمان پایان مشخص شده است. هدف پیدا کردن بینشی‌ین تعداد این فعالیت‌هاست که بتوانند از منبع استفاده کنند.

ورودی: برای هر فعالیت a_i (برای $i = 1 \dots n$) مقادیر زیر داده شده‌اند:

- زمان شروع: s_i
- زمان پایان: e_i

هدف: انتخاب حداقل فعالیت‌های ممکن.

مثال: هر فعالیت می‌تواند یک تقاضا برای استفاده از ملا سالن اجتماعات یک دانشگاه باشد.

ارائه یک روش ساده برای حل مسأله: من توان برای هر فعالیت‌های موجود، تمام فعالیت‌های دیگر که با آن در تضاد هستند را حذف نموده و این روش را نکرار کرد و تکلیف مجموعه فعالیت‌های ممکن را بدست آورد. در پایان بزرگ‌ترین مجموعه‌ی ممکن جواب است. الگوریتم بسیار کند و از مرتبه‌ی $O(n!)$ است چون همه‌ی جای‌گشتها را بررسی می‌کند.

روش حریصانه

در الگوریتم حریصانه بدنبال روشی برای انتخاب یک فعالیت مناسب هستیم که در صورتی که با انتخاب‌های تناقض نداشته باشد آنرا به صورت نهایی انتخاب کنیم. برای این کار می‌توانیم ترتیب انتخاب‌ها را براساس ترتیب اوله، طول فعالیت‌ها، زمان شروع با، زمان پایان از کوچک به بزرگ با مراعکس (احادم دهم، شخصی است که در دو مورد اول به شکل بر می‌خورد، مثلاً در مورد ترتیب انتخاب براساس طول فعالیت‌ها، اگر از دو فعالیت، کوچک‌ترین آن‌ها با دو فعالیت دیگر (که از هم مجزا هستند) اشتراک داشته باشد، ما فقط همان کوچک‌ترین فعالیت را می‌توانیم انتخاب کنیم؛ در صورتی که دو فعالیت دیگر جواب بدهی است. برای ترتیب‌های دیگر، با هر آخرين ترتیب تیز می‌توان مثال فرض کرد.

ساده‌بین، به نظر می‌رسد که اگر نهایت‌ها را به ترتیب پایان زمان‌شان انتخاب کنیم و پس از انتخاب یک فعالیت، همه‌ی فعالیت‌های منضاد با آن را حذف کنیم، جواب بدهی به دست می‌آید. البته درستی این الگوریتم حریصانه را باید اثبات کنیم.

مراحل کلی اثبات درستی یک الگوریتم حریصانه:

۱. اثبات می‌کنیم که بک راه حل بهینه وجود دارد که شامل اولین انتخاب الگوریتم بشهادی است.

فصل ۲ روش‌های طراحی الگوریتمها

۱۵۰

۲. با حذف انتخاب اولیه و انتخاب‌های متساد با آن یک زیرمسئله بددست می‌آید که باید آن را تبر به صورت بهینه و بهینه روش حل کنم.

در مسئله‌ی انتخاب فعالیت‌ها، ابتدا یک زیرمسئله‌ی کلی را تعریف می‌کیم و سپس به اثبات دقیق درستی الگوریتم ارائه شده می‌پردازیم. یک زیرمسئله شامل k عدد فعالیت است (در مسئله‌ی اصلی $k = 6$). بدون کم شدن از کلیت مسئله می‌توان فرض کرد که شماره فعالیت‌های یک زیرمسئله بهمان ترتیب زمان‌های پایان آن‌ها (از کوچک به بزرگ) است. یعنی فرض می‌کیم که $t_1 \leq t_2 \leq \dots \leq t_k$.

لم ۷. یک راه حل بهینه برای مسئله وجود دارد که شامل کار ۱ است.

اثبات. برهان خلف: فرض A: یک راه حل بهینه برای مسئله داده شده است که t_1 متعلق به آن نیست. فرض کنید، فعالیت با کمترین زمان پایان در A است. بنابراین فرض داریم: $t_1 \leq t_2$. حال اگر فعالیت t_1 را از A حذف کنیم و به جای آن t_2 را فراز دهیم، مجموعه کلیه فعالیت‌ها حاصل نیز یک جواب برای مسئله است، چون t_2 حداقل می‌تواند با t_1 در A اشتراک داشته باشد و با بقیه فعالیت‌ها اشتراکی ندارد. چون تعداد فعالیت‌های موجود در A و در جواب جدید برابرند، این جواب نیز بهینه است. پس ما یک جواب بهینه شامل t_1 پیدا کردیم.

با انتخاب t_1 و سپس حذف کلیه فعالیت‌هایی که با آن اشتراک دارند، یک زیرمسئله‌ی کوچک‌تر به دست می‌آید که آن را تبر بهینه روش حل می‌کیم. بنابراین این الگوریتم به یک مرتب‌سازی اولیه بیاز دارد و بقیه‌ی الگوریتم از مرتبه‌ی $O(n^2)$ انجام می‌شود. پس پیچیدگی الگوریتم $O(n^2)$ است.

مثال: الگوریتم پیشنهادی بر روی زاده‌های نمونه به صورت زیر عمل کرده است.

فعالیت	زمان شروع (t_i)	زمان پایان (t_f)	انتخاب باحدف
۱	۱	۳	۱
۲	۳	۴	۲
۳	۴	۵	۳
۴	۵	۶	۴
۵	۶	۷	۵
۶	۷	۸	۶
۷	۸	۹	۷
۸	۹	۱۰	۸
۹	۱۰	۱۱	۹
۱۰	۱۱	۱۲	۱۰
۱۱	۱۲	۱۳	۱۱
۱۲	۱۳	۱۴	۱۲
۱۳	۱۴	۱۵	۱۳
۱۴	۱۵	۱۶	۱۴
۱۵	۱۶	۱۷	۱۵
۱۶	۱۷	۱۸	۱۶
۱۷	۱۸	۱۹	۱۷
۱۸	۱۹	۲۰	۱۸
۱۹	۲۰	۲۱	۱۹
۲۰	۲۱	۲۲	۲۰
۲۱	۲۲	۲۳	۲۱
۲۲	۲۳	۲۴	۲۲
۲۳	۲۴	۲۵	۲۳
۲۴	۲۵	۲۶	۲۴
۲۵	۲۶	۲۷	۲۵
۲۶	۲۷	۲۸	۲۶
۲۷	۲۸	۲۹	۲۷
۲۸	۲۹	۳۰	۲۸
۲۹	۳۰	۳۱	۲۹
۳۰	۳۱	۳۲	۳۰
۳۱	۳۲	۳۳	۳۱
۳۲	۳۳	۳۴	۳۲
۳۳	۳۴	۳۵	۳۳
۳۴	۳۵	۳۶	۳۴
۳۵	۳۶	۳۷	۳۵
۳۶	۳۷	۳۸	۳۶
۳۷	۳۸	۳۹	۳۷
۳۸	۳۹	۴۰	۳۸
۳۹	۴۰	۴۱	۳۹
۴۰	۴۱	۴۲	۴۰
۴۱	۴۲	۴۳	۴۱
۴۲	۴۳	۴۴	۴۲
۴۳	۴۴	۴۵	۴۳
۴۴	۴۵	۴۶	۴۴
۴۵	۴۶	۴۷	۴۵
۴۶	۴۷	۴۸	۴۶
۴۷	۴۸	۴۹	۴۷
۴۸	۴۹	۵۰	۴۸
۴۹	۵۰	۵۱	۴۹
۵۰	۵۱	۵۲	۵۰
۵۱	۵۲	۵۳	۵۱
۵۲	۵۳	۵۴	۵۲
۵۳	۵۴	۵۵	۵۳
۵۴	۵۵	۵۶	۵۴
۵۵	۵۶	۵۷	۵۵
۵۶	۵۷	۵۸	۵۶
۵۷	۵۸	۵۹	۵۷
۵۸	۵۹	۶۰	۵۸
۵۹	۶۰	۶۱	۵۹
۶۰	۶۱	۶۲	۶۰
۶۱	۶۲	۶۳	۶۱
۶۲	۶۳	۶۴	۶۲
۶۳	۶۴	۶۵	۶۳
۶۴	۶۵	۶۶	۶۴
۶۵	۶۶	۶۷	۶۵
۶۶	۶۷	۶۸	۶۶
۶۷	۶۸	۶۹	۶۷
۶۸	۶۹	۷۰	۶۸
۶۹	۷۰	۷۱	۶۹
۷۰	۷۱	۷۲	۷۰
۷۱	۷۲	۷۳	۷۱
۷۲	۷۳	۷۴	۷۲
۷۳	۷۴	۷۵	۷۳
۷۴	۷۵	۷۶	۷۴
۷۵	۷۶	۷۷	۷۵
۷۶	۷۷	۷۸	۷۶
۷۷	۷۸	۷۹	۷۷
۷۸	۷۹	۸۰	۷۸
۷۹	۸۰	۸۱	۷۹
۸۰	۸۱	۸۲	۸۰
۸۱	۸۲	۸۳	۸۱
۸۲	۸۳	۸۴	۸۲
۸۳	۸۴	۸۵	۸۳
۸۴	۸۵	۸۶	۸۴
۸۵	۸۶	۸۷	۸۵
۸۶	۸۷	۸۸	۸۶
۸۷	۸۸	۸۹	۸۷
۸۸	۸۹	۹۰	۸۸
۸۹	۹۰	۹۱	۸۹
۹۰	۹۱	۹۲	۹۰
۹۱	۹۲	۹۳	۹۱
۹۲	۹۳	۹۴	۹۲
۹۳	۹۴	۹۵	۹۳
۹۴	۹۵	۹۶	۹۴
۹۵	۹۶	۹۷	۹۵
۹۶	۹۷	۹۸	۹۶
۹۷	۹۸	۹۹	۹۷
۹۸	۹۹	۱۰۰	۹۸
۹۹	۱۰۰	۱۰۱	۹۹
۱۰۰	۱۰۱	۱۰۲	۱۰۰
۱۰۱	۱۰۲	۱۰۳	۱۰۱
۱۰۲	۱۰۳	۱۰۴	۱۰۲
۱۰۳	۱۰۴	۱۰۵	۱۰۳
۱۰۴	۱۰۵	۱۰۶	۱۰۴
۱۰۵	۱۰۶	۱۰۷	۱۰۵
۱۰۶	۱۰۷	۱۰۸	۱۰۶
۱۰۷	۱۰۸	۱۰۹	۱۰۷
۱۰۸	۱۰۹	۱۱۰	۱۰۸
۱۰۹	۱۱۰	۱۱۱	۱۰۹
۱۱۰	۱۱۱	۱۱۲	۱۱۰
۱۱۱	۱۱۲	۱۱۳	۱۱۱
۱۱۲	۱۱۳	۱۱۴	۱۱۲
۱۱۳	۱۱۴	۱۱۵	۱۱۳
۱۱۴	۱۱۵	۱۱۶	۱۱۴
۱۱۵	۱۱۶	۱۱۷	۱۱۵
۱۱۶	۱۱۷	۱۱۸	۱۱۶
۱۱۷	۱۱۸	۱۱۹	۱۱۷
۱۱۸	۱۱۹	۱۲۰	۱۱۸
۱۱۹	۱۲۰	۱۲۱	۱۱۹
۱۲۰	۱۲۱	۱۲۲	۱۲۰
۱۲۱	۱۲۲	۱۲۳	۱۲۱
۱۲۲	۱۲۳	۱۲۴	۱۲۲
۱۲۳	۱۲۴	۱۲۵	۱۲۳
۱۲۴	۱۲۵	۱۲۶	۱۲۴
۱۲۵	۱۲۶	۱۲۷	۱۲۵
۱۲۶	۱۲۷	۱۲۸	۱۲۶
۱۲۷	۱۲۸	۱۲۹	۱۲۷
۱۲۸	۱۲۹	۱۳۰	۱۲۸
۱۲۹	۱۳۰	۱۳۱	۱۲۹
۱۳۰	۱۳۱	۱۳۲	۱۳۰
۱۳۱	۱۳۲	۱۳۳	۱۳۱
۱۳۲	۱۳۳	۱۳۴	۱۳۲
۱۳۳	۱۳۴	۱۳۵	۱۳۳
۱۳۴	۱۳۵	۱۳۶	۱۳۴
۱۳۵	۱۳۶	۱۳۷	۱۳۵
۱۳۶	۱۳۷	۱۳۸	۱۳۶
۱۳۷	۱۳۸	۱۳۹	۱۳۷
۱۳۸	۱۳۹	۱۴۰	۱۳۸
۱۳۹	۱۴۰	۱۴۱	۱۳۹
۱۴۰	۱۴۱	۱۴۲	۱۴۰
۱۴۱	۱۴۲	۱۴۳	۱۴۱
۱۴۲	۱۴۳	۱۴۴	۱۴۲
۱۴۳	۱۴۴	۱۴۵	۱۴۳
۱۴۴	۱۴۵	۱۴۶	۱۴۴
۱۴۵	۱۴۶	۱۴۷	۱۴۵
۱۴۶	۱۴۷	۱۴۸	۱۴۶
۱۴۷	۱۴۸	۱۴۹	۱۴۷
۱۴۸	۱۴۹	۱۵۰	۱۴۸
۱۴۹	۱۵۰	۱۵۱	۱۴۹
۱۵۰	۱۵۱	۱۵۲	۱۵۰
۱۵۱	۱۵۲	۱۵۳	۱۵۱
۱۵۲	۱۵۳	۱۵۴	۱۵۲
۱۵۳	۱۵۴	۱۵۵	۱۵۳
۱۵۴	۱۵۵	۱۵۶	۱۵۴
۱۵۵	۱۵۶	۱۵۷	۱۵۵
۱۵۶	۱۵۷	۱۵۸	۱۵۶
۱۵۷	۱۵۸	۱۵۹	۱۵۷
۱۵۸	۱۵۹	۱۶۰	۱۵۸
۱۵۹	۱۶۰	۱۶۱	۱۵۹
۱۶۰	۱۶۱	۱۶۲	۱۶۰
۱۶۱	۱۶۲	۱۶۳	۱۶۱
۱۶۲	۱۶۳	۱۶۴	۱۶۲
۱۶۳	۱۶۴	۱۶۵	۱۶۳
۱۶۴	۱۶۵	۱۶۶	۱۶۴
۱۶۵	۱۶۶	۱۶۷	۱۶۵
۱۶۶	۱۶۷	۱۶۸	۱۶۶
۱۶۷	۱۶۸	۱۶۹	۱۶۷
۱۶۸	۱۶۹	۱۷۰	۱۶۸
۱۶۹	۱۷۰	۱۷۱	۱۶۹
۱۷۰	۱۷۱	۱۷۲	۱۷۰
۱۷۱	۱۷۲	۱۷۳	۱۷۱
۱۷۲	۱۷۳	۱۷۴	۱۷۲
۱۷۳	۱۷۴	۱۷۵	۱۷۳
۱۷۴	۱۷۵	۱۷۶	۱۷۴
۱۷۵	۱۷۶	۱۷۷	۱۷۵
۱۷۶	۱۷۷	۱۷۸	۱۷۶
۱۷۷	۱۷۸	۱۷۹	۱۷۷
۱۷۸	۱۷۹	۱۸۰	۱۷۸
۱۷۹	۱۸۰	۱۸۱	۱۷۹
۱۸۰	۱۸۱	۱۸۲	۱۸۰
۱۸۱	۱۸۲	۱۸۳	۱۸۱
۱۸۲	۱۸۳	۱۸۴	۱۸۲
۱۸۳	۱۸۴	۱۸۵	۱۸۳
۱۸۴	۱۸۵	۱۸۶	۱۸۴
۱۸۵	۱۸۶	۱۸۷	۱۸۵
۱۸۶	۱۸۷	۱۸۸	۱۸۶
۱۸۷	۱۸۸	۱۸۹	۱۸۷
۱۸۸	۱۸۹	۱۹۰	۱۸۸
۱۸۹	۱۹۰	۱۹۱	۱۸۹
۱۹۰	۱۹۱	۱۹۲	۱۹۰
۱۹۱	۱۹۲	۱۹۳	۱۹۱
۱۹۲	۱۹۳	۱۹۴	۱۹۲
۱۹۳	۱۹۴	۱۹۵	۱۹۳
۱۹۴	۱۹۵	۱۹۶	۱۹۴
۱۹۵	۱۹۶	۱۹۷	۱۹۵
۱۹۶	۱۹۷	۱۹۸	۱۹۶
۱۹۷	۱۹۸	۱۹۹	۱۹۷
۱۹۸	۱۹۹	۲۰۰	۱۹۸
۱۹۹	۲۰۰	۲۰۱	۱۹۹
۲۰۰	۲۰۱	۲۰۲	۲۰۰
۲۰۱	۲۰۲	۲۰۳	۲۰۱
۲۰۲	۲۰۳	۲۰۴	۲۰۲
۲۰۳	۲۰۴	۲۰۵	۲۰۳
۲۰۴	۲۰۵	۲۰۶	۲۰۴
۲۰۵	۲۰۶	۲۰۷	۲۰۵
۲۰۶	۲۰۷	۲۰۸	۲۰۶
۲۰۷	۲۰۸	۲۰۹	۲۰۷
۲۰۸	۲۰۹	۲۱۰	۲۰۸
۲۰۹	۲۱۰	۲۱۱	۲۰۹
۲۱۰	۲۱۱	۲۱۲	۲۱۰
۲۱۱	۲۱۲	۲۱۳	۲۱۱
۲۱۲	۲۱۳	۲۱۴	۲۱۲
۲۱۳	۲۱۴	۲۱۵	۲۱۳
۲۱۴	۲۱۵	۲۱۶	۲۱۴
۲۱۵	۲۱۶	۲۱۷	۲۱۵
۲۱۶	۲۱۷	۲۱۸	۲۱۶
۲۱۷	۲۱۸	۲۱۹	۲۱۷
۲۱۸	۲۱۹	۲۲۰	۲۱۸
۲۱۹	۲۲۰	۲۲۱	۲۱۹
۲۲۰	۲۲۱	۲۲۲	۲۲۰
۲۲۱	۲۲۲	۲۲۳	۲۲۱
۲۲۲	۲۲۳	۲۲۴	۲۲۲
۲۲۳	۲۲۴	۲۲۵	۲۲۳
۲۲۴	۲۲۵	۲۲۶	۲۲۴
۲۲۵			

۵.۴ روش حرسانه در طراحی الگوریتمها

مسئلهی خرد کردن پول

من خواهیم ۸ تومان را سکه‌های ۲، ۱ و ۰ تومانی خرد کیم که مجموع تعداد سکه‌هایی که استفاده می‌کیم حداقل شود. لزی هر یک از این سکه‌ها بتداد زیادی در اختیار داریم. این الگوریتم معمولاً برای این مسئله ارائه می‌شود:

در ابتدا هرچه توابیم سکه‌های ۰ تومانی برمی‌داریم تا جایی که مقدار باقی مانده از ۵ تومان کمتر شود.

سپس آنقدر سکه‌ی ۲ تومانی برمی‌داریم تا مقدار باقی مانده از ۵ تومان کمتر شود و در نهایت مقدار باقی مانده را از سکه‌های یک تومانی برمی‌داریم.

این یک الگوریتم حرسانه است، چون در مرحله پنجم‌ترین سکه‌ای که می‌تواند انتخاب می‌کند، آنرا این الگوریتم همواره جواب بهمنه را می‌دهد؟ جواب این سوال برای این سکه‌ها مشتب است. اثبات این امر به تصریف و اگذار می‌شود، برای مقادیر دیگری از سکه‌ها (برابر ۱، ۰، ...) هم این الگوریتم درست کار می‌کند. اما اگر به جای سکه‌های ۰، ۱ و ۰ تومانی سکه‌های دیگری در اختیار داشتهیم الگوریتم لزوماً درست عمل نمی‌کند. مثلًا اگر به جای سکه‌ی ۲ تومانی سکه‌ی ۲ تومانی داشتهیم، الگوریتم ۸ تومان را یک سکه‌ی ۵ تومانی و سه سکه‌ی ۱ تومانی (مجموعاً ۲ سکه) خود می‌کند، در حالی که فقط با دو سکه ۳ تومانی بهتر خرد می‌شود. روش است که در حالت کلی این مسئله همان مسئله کوله‌پشتی است که ارزش هر بار ۱ و تعداد هر نوع بار بینهایت است. این مسئله را بیلا حل کردیم.

مسئلهی کوله‌پشتی با بارهای قابل تقسیم

مسئلهی کوله‌پشتی در صورتی که بارها را بتوابع تقسیم کنیم (fractional knapsack problem) بیزرا حل دارد.

فرض کنید N عدد بار داده شده است که وزن و ارزش بار i ام به ترتیب برابر w_i و c_i است و می‌خواهد i کوله‌پشتی به اندازه‌ی M را با این بارها پر کنیم به طوری که ارزش بارهای انتخابی بیشترین شود. در این مورد مجاز است که یک بار را به نسبت دل خواه به دو قسمت تقسیم کنیم.

الگوریتم ساده‌ی زیر این کار را انجام می‌دهد:

بارها را به ترتیب ارزش در واحد وزن مرتب می‌کنیم. یعنی فرض می‌کنیم $\frac{c_1}{w_1} \geq \frac{c_2}{w_2} \geq \dots \geq \frac{c_n}{w_n}$. بارها را به همین ترتیب مورد بررسی قرار می‌دهیم. اگر بار i می‌تواند کاملاً در کوله‌پشتی باقی مانده قرار بگیرد این کار را می‌کنیم و مراجع بار بعدی می‌ردم. اگر وزن بار i ام بیشتر از مقدار باقی مانده از کوله‌پشتی است، این بار را به نسبت موردنظر تقسیم می‌کنیم تا کوله‌پشتی کاملاً پرشود. واضح است که این الگوریتم درست کار می‌کند. (اثبات دقیق را به خودتان وا می‌گذاریم.)

۴.۵.۴ مسائل زمان‌بندی

در حالت کلی این مسائل NP-Complete هستند اما برای برخی از آن‌ها با روش حرسانه راه حل دقیق دارند.

حالات ماده

بک سرویس دهنده و چند کار (job) آماده گرفتن سرویس داده شده‌اند. زمان موردنیاز هر کار داده شده است. می‌خواهیم به تمام این کارها به ترتیبی سرویس دهیم و یک پارامتر سیستم را بهینه سازیم. پارامترهای سیستم می‌توانند:

۱۰۱

متوسط زمان پاسخ، حد اکثر با متوسط زمان انتظار باشد. شالی از این مسئله را در صفحه‌های مختلف در عمل می‌توان دید. چنین صفحه‌هایی هم در سیستم عامل داریم و سیاری از الگوریتم‌های زمان‌بندی در اینجا کاربرد دارند.

به بیان دیگر، یک پردازنده، n عدد کار، d_1, d_2, \dots, d_n در زمان صفر داده شده‌اند، به طوری که زمان موردنیاز کار i برابر d_i است (service time). می‌خواهیم یک زمان‌بندی پردازنده‌ای ترتیب اجرا (scheduling) کرده تا متوسط زمان پاسخ (average response time) حداقل شود. زمان پاسخ کار i برابر است با مدت زمان انتظار این کار پلاس زمان پاسخ d_i .

می‌توان به طور شهودی دید که اگر کارها را به ترتیب افزایشی زمان سرویس آن‌ها در صفحه قرار دهیم، مجموع زمان‌های پاسخ کمینه می‌شود. برای اثبات، فرض می‌کنیم که $d_1 \leq d_2 \leq \dots \leq d_n$.

آنکه برای مسئله زمان‌بندی بهینه‌ی یک پردازنده، یک راه حل بهینه وجود دارد که در آن اولین کاری که از پردازنده سرویس می‌گیرد، کار 1 است.

این اثبات با پرهان خلف: فرض کنید که اولین کار i باشد ($i > 1$). بنابراین کار i بین کارهای دیگر در صفحه قرار می‌گیرد. فرض کنید که j کاری است که قبل از i انجام می‌شود. نشان می‌دهیم که با تعویض دو کار i و j مجموع زمان‌های پاسخ بدتر نمی‌شود.

بدینهین است که تعویض دو کار مجاور تأثیری بر زمان پاسخ کارهای دیگر ندارد. بنابراین فقط مجموع زمان‌های پاسخ کارهای 1 و j مینم است. اگر D زمان اتمام کار جلوی کار j باشد، مجموع زمان‌های پاسخ این دو کار در حالت اول برآور است با $d_1 + d_j + d_2 + \dots + d_{j-1} = 2D + TD_j$. و در حالت دوم برآور است با $d_1 + d_j + d_2 + \dots + d_{j-1} = 2D + TD_j + d_i$.

بدینهین است که $A - B = d_j - d_i \geq 0$. ($A = D + d_1 + D + d_2 + \dots + d_{j-1}$ و $B = D + d_1 + d_2 + \dots + d_{j-1} + d_i$) با این ترتیب، روش است که کار i را من توان همراه با کار جلوی آن تعویض کرد و هر بار عبارت موردنظر بهتر شود (با بدتر شود) تا آن جا که این کار اولین کار صفحه بخواهد. سپس یک راه حل بهینه وجود دارد که، اولین کار سرویس گیرنده است.

با اثبات دو روش یا چند پردازنده نیز به همین ترتیب عمل می‌کنیم. اگر در مسئله فوق روابط پیش‌نبایزی وجود داشته باشد، در حالت کلی مسئله NP-Complete است ولی برای یک با دو پردازنده راه حل چندجمله‌ای وجود دارد.

زمان‌بندی کارها با جنبه‌ی تأخیر

ا) کار با شماره‌های $1 \dots n$ دارد، شده‌اند که زمان اجرای هر کدام 1 واحد زمان است. d_i مهلت انجام کار i است و اگر این کار بعد از زمان i به اتمام رسید جریمه‌ای برای i به آن تعليق می‌گیرد. فرض می‌کنیم که مهلات‌ها اعتماد صحیح هستند. هدف تعیین یک زمان‌بندی برای اجرای همه این کارهایست به طوری که مجموع جریمه‌ها کمینه شود.

مشاهدهای ما از مسئله: فرض می‌کنیم زمان‌بندی بهینه را داریم. این زمان‌بندی را به صورت شکل زیر را در نظر می‌گیریم که در آن هر کار در یکی از مجموعه‌ای زمانی بداندای 1 واحد تخصیص داده شده است.

جون به هر کار با تأخیر فقط یک مقدار جریمه تعليق می‌گیرد و میزان تأخیر در این جریمه تأخیر ندارد، روش است که می‌توان در زمان‌بندی بهینه (یا هر زمان‌بندی داده شده) همه کارهای با تأخیر (late tasks) را در انتهای زمان‌بندی و کارهای بدون تأخیر (early tasks) (را در ابتدای آن انجام داد) آن که در میزان جریمه تغییری داده شود. این کار با تکرار عمل محاذ جایی یک کار بدون تأخیر، مانند i یا یک کار با تأخیر که بالا صalte قبل از i انجام داد.

۵.۴ روش حرصانه در طراحی الگوریتمها

همچنین در زمان بندی داده شده می توان کارهای بدون تأخیر را بر حسب مهلت هایشان (از کوچک به بزرگ) مرتب کرد. زیر هر دو کار بدون تأخیر و مجاز و $i > k$ که با پالانفسله بعد از i اجرا شود، ولی داشته باشیم $i < k$ را می توان با هم جایه ساز کرد و هر دو کار هنوز قبل از مهلت شان اجرا شوند (با توجه به این که i قبل از زمان k اجرا می شود و $i + 1 \geq k$ پس با اجرای $i + 1$ یک واحد زمانی بیشتر هنوز آن کار قبل از مهلتش اجرا می شود، اجرای رویدن i مشکلی ایجاد نمی کند).

با این مشاهدات الگوریتم حرصانه زیر را برای این مسئله پیشنهاد می کنیم:

۱. کارها را به ترتیب جریمه هایشان از بزرگ به کوچک مورد بررسی قرار می دهیم. فرض کنید کار i انتخاب کرده ایم.

۲. آخرین بازه زمانی ای را پیدا می کنیم که i را بتوانیم در آن قبل از مهلتش انجام دهیم. برای این کار از بازه $i - d_i$ تا i آغاز و بازهای سمت چپ آنرا به ترتیب راست به جه مورده بررسی قرار می دهیم. سمت راست ترین بازه زمانی خالی محل قرار گرفتن کار i است.

۳. اگر بازه زمانی خالی برای اجرای i بدون تأخیر برای کار i پیدا نشود، این کار با تأخیر انجام می شود و بعد از تعیین همه کارهای بدون تأخیر زمان بندی می شود.

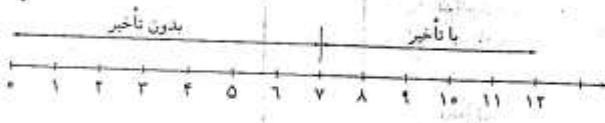
لم ۹. الگوریتم حرصانه از اندشه بگ جواب بهمن برای مسئله فوق بددست می آورد.

البات. برای اثبات، باید زیر مسئله را به درستی تعریف کنیم. فرض می کنیم $W_1 \geq \dots \geq W_k \geq W_{k+1}$. زیر مسئله شامل کارهای i تا n است که باید از یک بازه زمانی ثابتی قرار داده شوند. با توجه به این که قابل تکلیف کارهای i تا n مشخص شده است، فرض می کنیم که معدادی (حداکثر m) از بازه ها قابل پرشدن و قابل استفاده نیستند.

نشان می دهیم که اگر امکان اجرای کار i قبل از مهلتش باشد، بگ جواب بهمن وجود دارد که در آن i بدون تأخیر اجرا می شود.

فرض کنید که بگ جواب بهمن وجود دارد که در آن i با تأخیر اجرا می شود. کاری که در بازه $i - d_i$ تا i قرار دارد را را بنامید. می دانیم که $i > z_k$ پس i بدون تأخیر اجرا می شود. با جایه جایی i و z_k در زمان بندی جدید i با تأخیر اجرا می شود که با توجه به این که می دانیم $z_k \geq i - d_i$ مجموع جرمیه ها پرشدنی شود. اگر i هم بدون تأخیر اجرا شود، وضعیت بهمن می شود.

با توجه i حداکثر یک بازه زمانی جدید اشغال می شود و با کم شده این کار، بگ زیر مسئله کوچکتر ابعاد می شود که به همین روش حل می شود.



۱۰۳

γ	γ	α	β	γ	β	γ	α	β	γ	δ
γ	β	1	γ	β	γ	β	γ	β	γ	d_1
β	γ	γ	β	β	α	γ	γ	γ	β	B_1

مجموعه جریمه‌هایی که تعلق می‌گیرد ۵۰ است.

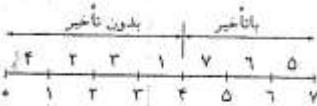
الگوریتم هافمن ۵.۵.۴

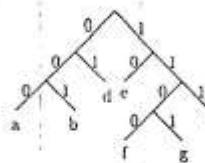
قابلی حاوی نویسه داده شده است. می خواهیم برای هر نویسه کدی طراحی کنیم به طوری که با استفاده از این کدها (به جای کدهای متلاطه بین قابل) اندازه‌ی فایل جدید (برحسب بیت) کمیته شود. تعداد بیت‌های کدهای طراحی شده، می تواند متفاوت و دلخواه باشد. این بهترین روش برای فشرده‌سازی فایل هاست با این شرط که هر نویسه کدگذاری شود. روش‌های دیگری وجود دارند که نیز رشته‌ها را به صورتی به کد تبدیل می کنند که ممکن است از این الگوریتم بهتر عمل کند.

در این مسئله با توجه به آن که فایل داده شده است، بنا بر این تعداد تکرار و با اختصار و قعو نویسه را دارم.
این الگوریتم را در مواردی مثلاً ارسال اطلاعات نزدیکی و به صورت «برخط» (online) بر روی شبکه در صورتی که اختصار و قعو نویسه ها با تقریب خوبی داشته باشند، بدون آن که کل فایل داده شده باشد بیشتر استفاده می شود. توجه کنید که تعریفها را، متأسفانه سه‌هاست مثلاً اسکریپت.

مسئله با صورت انتزاعی از قرار بر است. $\pi_{\text{نوسی}} = \frac{\pi_{\text{نا}}}{\pi_{\text{نا}} + \pi_{\text{ب}}}$ و احتمال وقوع بگیرای هر یک داده است
 $\pi_{\text{نوسی}} = \frac{\pi_{\text{نا}}}{\pi_{\text{نا}} + \pi_{\text{ب}}} = \frac{\pi_{\text{نا}}}{\sum_{i=1}^k \pi_i}$ می خواهیم به مر نوسی $\pi_{\text{نا}}$ کدی به طول k ، نسبت دهیم به طوری که متوسط طول کدها (یعنی $\sum_{i=1}^k i \pi_i$) کمتر شود.

روشن است که متوسط عمق برگ‌های درخت هامن همان متوسط قلوی گدھاست. گلورین هامن با دریافت احتمال و قوع نویسه‌ها، پک درخت هامن با حداقل متوسط عمق برگ‌ها ایجاد می‌کند. گلورین در ایستادن یک گره مرای هر نویسه‌ی ورودی ایجاد می‌کند و برای هر گره احتمال و قوع آن نویسه‌را





شکل ۴.۴: کدگذاری درست را می‌توان با یک درخت دودویی نمایش داد.

نیست می‌دهد. به صورت حریصانه هر بار در گرهی x و y با کمترین احتمال وقوع ($f(x) + f(y)$) را بیندازیم کند، این دو گره را فرزندان یک گرهی جدید (به نام مثلاً z) با احتمال وقوع $f(z) = f(x) + f(y)$ من کند. x و y را حذف و به جای آنها z را اضافه می‌کند. الگوریتم این کار را تکرار می‌کند تا این که تنها یک گره با احتمال وقوع ۱ که همان ریشه‌ی درخت است حاصل شود.

پیاده‌سازی الگوریتم به صورت زیر است:

```

Procedure Huffman (C)
Create an empty Q
for all c in C do
    Allocate_Node (x); f(x) <- f[c]
    Insert (x, Q)
For i:=1 to n-1 do
begin
    z <- Allocate_Node();
    x <- Left[z] <- Extract_min (Q);
    y <- Left[z] <- Extract_min (Q);
    f(z) <- f(x) + f(y);
    Insert (z, Q)
end;

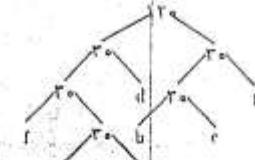
```

بهترین داده‌ساختار برای Q یک صف اولویت است که اعمال فوق را با $O(\lg n)$ انجام می‌دهد. پس الگوریتم فوق از $O(n \lg n)$ است.

مثال

قابلی به اندازه‌ی ۱۲۰ حاوی نویسه‌های زیر است.

۱۰۰



شکل ۴۵.۲: درخت هائین برای مثال.

موضع	فرانس تکرار
1	۰
۲	۱۳
۳	۱۷
۴	۲۵
۵	۲۷
۶	۳۰
۷	۳۱
۸	۳۴
۹	۳۵
۱۰	۳۶

اعمال اکبریت، تاریخ و حبیب نیل آنها را با خود از این طبقه ایزدی دوست نمی‌آمد. توجه کنید که

۶.۰.۴ الگوریتم خریصانه تقریبی برای مسئله بسته‌بندی

«شیء داده شده‌اند که حجم شیء آن برابر ۲ است، این عدد حقيقة بین صفر و یک هست. می‌خواهیم این اشیاء را در صندوقچه‌هایی که حجم هر کدام از آن‌ها برابر با ۱ است، بسته‌بندی کنیم به طوری که تعداد کل صندوقچه‌ها حداقل شود. این مسئله به نام بسته‌بندی (bin packing) معروف است. ساده‌ترین الگوریتم خریصانه که برای این مسئله معرفت می‌رسد، به این شیوه است:

برای قراردادن شیء i ، اگر یکی از صندوقچه‌هایی غیر خالی به اندازه i را در یک صندوقچه‌ی جدید قرار می‌دهیم
ام را بر آن قرار می‌دهیم و گزنه، آن را در یک صندوقچه‌ی جدید قرار می‌دهیم.

اما متأسفانه این الگوریتم در بسیاری از موارد اشتباه می‌کند. مثلاً اگر حجم اشیاء برابر $6/4, 3/4, 2/4, 1/4$ باشد، این الگوریتم این اشیاء را در ۳ صندوقچه بسته‌بندی می‌کند: در صندوقچه‌ی اول اشیاء با حجم‌های $1/4, 2/4$ و $3/4$ ، در صندوقچه‌ی دوم اشیاء با حجم‌های $3/4, 2/4, 1/4$ و $2/4$ و در صندوقچه‌ی سوم بقیه اشیاء با حجم‌های $6/4, 2/4$ و $3/4$ قرار گرفته است. درحالی که می‌توانستیم این اکار را با دو صندوقچه بین‌جامد دهیم، به این صورت که در صندوقچه‌ی اول اشیاء با حجم‌های $2/4, 1/4$ و $2/4$ و در صندوقچه دوم اشیاء با حجم‌های $3/4, 2/4$ و $2/4$ را فرازه دهیم، بدینم که این الگوریتم ممکن است عوایز بیشتر را به دست ندهد. اما اگر یک سوال پیش می‌آید: حواسی که این الگوریتم به ما می‌دهد تاچه حد نزدیک به جواب بیشتر است؟ قضیه‌ی حدودی به این پرسش پاسخ می‌دهد:

قضیه ۱۲. اگر حداقل تعداد صندوقچه‌های لازم برای بسته‌بندی اشیاء داده شده برابر با $OPT = OPT(2 \times \text{مسئله اینجام})$ باشد، الگوریتم فوق

این اشیاء را با حداقل $OPT \times 2$ صندوقچه انجام می‌دهد.

این اثبات در الگوریتم ما ممکن نیست که بین از یک صندوقچه باقی بماند که کمتر از نصف آن پر شده باشد. (جزئی) بنا بر این اگر $S = OPT$ می‌باشد، صندوقچه باقی بماند که کمتر از نصف آن پر شده باشد. هر دو از این اشیاء را بسته‌بندی کنیم، حداقل به S تا صندوقچه نیاز داریم. بنا بر این $S = OPT$ بنا بر این نات کردیم که الگوریتم ما حداقل از $2 \times OPT$ عدد صندوقچه استفاده می‌کند. (۲) بنا بر این الگوریتم خریصانه‌ای که برای این مسئله از این داده، زیاد هم بد نیست! جوابی که این الگوریتم به ما می‌دهد حداقل در برابر مقدار بیشتر است. در واقع با استفاده از یک روش پیچیده‌تر می‌توان ثابت کرد که این الگوریتم حداقل $2 \times OPT + 1$ برابر مقدار بیشتر را داشت می‌آورد. این مقاله‌ای هم می‌توان ساخت که الگوریتم ما دقیقاً $2 \times OPT + 1$ برابر مقدار بیشتر را داشت آورد. بنا بر این با استفاده از این الگوریتم می‌توانیم پیش از این پیش برویم.

حالا سعی می‌کنیم که کمی الگوریتم فوق را اصلاح کنیم و می‌خواهیم بیشتر بگیریم. بگذر کار عالملانه این است که اشیاء ای را به ترتیب نزولی حجم‌شان مرتب کنیم و می‌خواهیم الگوریتم را در مردم آن‌ها بکار ببریم. به این ترتیب می‌رسد که با این اصلاح، الگوریتم قادر بهتر شده است، اما هنوز خیلی وقت‌ها اشتباه می‌کند. ثابت شده است که جوابی که این الگوریتم به ما می‌دهد از $OPT + 2$ پیشتر است. البته این این موضوع چندان ساده نیست. (۳) این دو نکته معرفی شدند که مسئله بسته‌بندی یک مسئله NP-Complete است و انتظار از این بگردد راه حل حقیقی جمله‌ای برای آن بیهوده است.

۷.۰.۴ تمرین‌ها

۱. نایت کنید که الگوریتم خود کردن پول اگر سکه های موجود ۱، ۰ و ۵ تومانی باشد، جواب بهینه را بدست می آورد.
۲. نایت کنید که اگر سکه های موجود ۱، ۰، و ۵ تومانی باشند الگوریتم خود کردن پول درست کار می کند.
۳. فرض کنید که سکه هایی با ارزش نومان وجود دارند و از مر کدام از آنها نیز به تعداد نامحدودی در دسترس داریم. می خواهیم با استفاده از این سکه ها ۷ تومان پول را خرد کنیم به طوری که تعداد سکه هایی که استفاده شده باشند را برابر با حداقل تعداد سکه های لازم برای خرد کرد ۷ تومان پول می گیریم. نایت کنید که با استفاده از روابط پک الگوریتم برای خرد کردن پول طراحی کنید.
۴. در مسئله کوله پشتی، فرض کنید که این شرط را داریم که: ترتیب بسته ها و قسم که بر حسب وزن شان به طور معکوسی مرتب شوند. همان ترتیبی است که اگر آنها بر حسب ارزش شان به طور نزولی مرتب کنیم. نایت کنید که اگر این ضرط برقرار باشد، الگوریتم حریصانه درست کار می کند.
۵. در مسئله کوله پشتی، فرض کنید که « نوع جنس داریم که وزن و ارزش جنس قام به ترتیب برابر است. از هر کدام از این اجنس نیز فقط یک عدد موجود است. و از این صورت تعریف می کنیم: اگر تراها ک ا نوع جنس اول، دوم... و kام را در اختیار داشته باشیم، مقدار جداگانه ارزش اجنسی که می توان از این این k جنس انتخاب کرد و در کوله پشتی با ترتیب W جاده باشد. رابطه‌ی پارگشتن زیر برای نایت کنید و با استفاده از آن الگوریتم برای حل مسئله کوله پشتی صفر و پک باید:
۶. فرض کنید که در پک گراف و زندگانیکترین زیردرخت فرآگیر را بدست آوردهایم. حالا می خواهیم زیردرخت فرآگیری از این گراف را بدست آوریم که پس از کوچکترین زیردرخت فرآگیر، از بقیه‌ی زیردرخت‌های فرآگیر این گراف کم وزن شود. به عبارت دیگر می خواهیم زیردرخت فرآگیری را بدست آوریم که از نظر کم وزن بودن در مرتبه‌ی دوم قرار گرفته است. الگوریتم برای حل این مسئله باید:
۷. مسئله برمی‌نماید چند پردازنده (Multiprocessor Scheduling) به این صورت مطلع می شود: فرض کنید k+1 کامپیوتر داریم که سرعت عملی آنها باهم برابر است. می خواهیم « تا برای این کامپیوتراها اجرای کمین، زمان اخراج برنامه‌ی iام بر روی هر کدام از این کامپیوتراها برایر باشید. می خواهیم ببینیم که باید هر کدام از برنامه‌های اخراج کدام کامپیوتر را اجرا کنیم که زمان اخراجی ممکن است این این لحظه‌ای که اجزای تمام برنامه‌ها به پایان رسیده است، حداقل باشند. پک الگوریتم حریصانه برای این مسئله پیشنهاد می کنیم: برنامه‌ها را به ترتیب نزولی زمان اخراج‌ستان مرتب می کنیم. میں اولین برنامه را به اولین کامپیوتر، دویم را به دویمن کامپیوتر... و k+1ین برنامه را به k+1ین کامپیوتر اختصاص می دهیم. میں از این اولین کامپیوتری که کارش تمام شد، اولین برنامه‌ای که هموز می‌بین کامپیوتری اختصاص داده شده است را به آن کامپیوتر می دهیم. این کار را تا جایی ادامه می دهیم که تمام برنامه‌ها بر روی کامپیوتراها اخراج شوند. اولاً نایت کنید که الگوریتم فوق درست است! یعنی مثلاً نقصی پیدا کنید که الگوریتم فوق نتواند جواب بهینه را برای آن بدست آوره. ثانياً نایت کنید که اگر جواب بهینه برای مسئله برایر با T باشد، الگوریتم فوق جوابی برای این مسئله بدست می آورد که زمان آن از T+T بیشتر است.
۸. پک گراف k+1 دارد، شده است. می خواهیم راسهای این گراف را رنگ آمیزی کنیم به طوری که شرایط زیر برقرار باشند: a) هیچ دو راس مجاوری هم رنگ نباشند. b) - تعداد رنگها که استفاده می کنیم حداقل باشد. ساده ترین الگوریتم حریصانه‌ای که می توان برای این مسئله پیشنهاد کرد به این صورت است: ابتدا پک ترتیب «لحواء» مانند راس رنگ گراف در تظریم گیریم. راس را با رنگ شماره‌ی i رنگ آمیزی می کنیم و پس از آن برای از این رنگی که در مجاورت این راس ظاهر شده است. اگر جنین رنگی موجود نبود. پک رنگ جدید را به مجموعه

۵.۴ روش حرصانه در طراحی الگوریتمها

ی رنگها اضافه می کنیم و از آن برای رنگ آمیزی را استفاده می کنیم. به سادگی می توانید مثال نقضی برای الگوریتم فوق بیابد. (در واقع هیچ الگوریتم حرصانه ای وجود ندارد که جواب بهینه را برای این مسئله بپدا کن). را برایر با تعداد رنگهایی می گیریم که الگوریتم فوق برای رنگ آمیزی گراف از آن استفاده می کند، اگر ترتیب اولیه ۰ را برای وقوس انتخاب کرده باشیم، همچنین (G) را برایر با حداقل تعداد رنگهایی که برای رنگ آمیزی ۰ را برای وقوس انتخاب کرده باشیم، می گیریم نات بگذارد که: الف - برای هر گراف G، ترتیب ۰ وجود دارد که شود. به عبارت دیگر گراف G لازم است می گیریم نات بگذارد که: الف - برای هر گراف G، ترتیب ۰ وجود دارد که شود. به عبارت دیگر در هر گراف اگر ترتیب اولیه ۰ ترتیب مناسب باشد، الگوریتم درست عمل می کند. ب - برای هر عدد Δ حقيقة ی گراف G و ترتیب ۰ برای وقوس آن وجود دارد که نسبت از بیشتر شود. به عبارت دیگر الگوریتم حرصانه می تواند به مقادیر دلخواهی اشتباه کند.

۹. گراف بدون جهت G داده شده است. مسئله پوشش راسی (Vertex-Cover) به این صورت مطرح می شود: کوچکترین زیرمجموعه \mathcal{C} از مجموعه \mathcal{V} گراف را پیدا کنید که برای هر یک از یالهای گراف، حداقل یکی از در سر این یال در مجموعه \mathcal{C} باشد. پک الگوریتم مکافعه ای ساده برای پیدا کردن کوچکترین پوشش راسی به این صورت است: (۱) \mathcal{C} را ساوه مجموعه \mathcal{V} تهی و E^* را ساوه مجموعه بالهای گراف فرازد. (۲) بال دلخواه \mathcal{U} از مجموعه \mathcal{V} را انتخاب کن (۳) در اس \mathcal{U} و $\mathcal{V} \setminus \mathcal{U}$ را به مجموعه \mathcal{C} اضافه کن (۴) تمامی بالهای \mathcal{C} را از دوسران \mathcal{U} یا $\mathcal{V} \setminus \mathcal{U}$ باشد از مجموعه \mathcal{E}^* حذف کن. (۵) اگر \mathcal{E}^* مجموعه تهی نیست، به مرحله ۲ برگرد. (۶) مجموعه \mathcal{C} را به عنوان خروجی الگوریتم برگردان.

ایندا مثال نقضی پیدا کنید که الگوریتم فوق درست کار نکند و میتواند می تواند به عبارت دیگر الگوریتم فوق می دهد حداقل تعداد رنگهایی مخصوص دارد.

۱۰. راندهایی من خواهد با ماشین از یک شهر به شهر دیگری سفر کند، ماشین با ایک بنزین بر می تواند ۱۱ کیلومتر حرکت کند. در مسیر مسافت این راننده n بین بنزین و وجود دارد. (یعنی بنزین اول در شهر مبدأ و بین بنزین k در شهر مقصد است) فاصله ای بین بنزین i تا بنزین $i+1$ نام برآورده است. اگر اعداد a_1, a_2, \dots, a_n داده شده باشد، می خواهیم حداقل تعداد بین بنزین هایی را پیدا کنیم که راننده می تواند با پرگرفتن باک بنزین ماشینش در این پیمایشها سافرت خود را انجام دهد پک الگوریتم حرصانه برای حل این مسئله پیشنهاد کند و آن را نات بگند.

۱۱. نقطعه با طول های و... و روی محوز طول ها داده شده است. الگوریتمی طراحی کنید که کمترین تعداد بازه های بد طول ۱ را پیدا کند که تمامی این نقاط را پوشاند. درستی الگوریتم خود را نات بگند.

۱۲. در یک فروشگاه n مشتری منتظر هستند تا کارشان انجام شود. کار مشتری شماره i (با اندازه i) دفعه طول می کشد. پک نفر کارمند متعددی انجام کار این مشتری هاست و در هر لحظه می تواند کار فقط پک مشتری را انجام دهد. می خواهیم بینیم که کارمند این فروشگاه باید کار این مشتری ها را به چه ترتیب انجام دهد تا مجموع زمان معطل شدن این مشتری ها کمینه شود. (زمان معطل شدن پک مشتری بر اساس زمانی است که انجام کار او به اتمام می رسد). پک الگوریتم حرصانه برای حل این مسئله پیدا کنید و میس درستی الگوریتم خود را اثبات نماید.

۱۳. ۱۱. برآمده با طول های a_1, a_2, \dots, a_n یک نوار مغناطیسی ذخیر شوند. می دانیم که اختلال این a_i بخواهیم برآمده ای شماره i را از روی نوار بخواهیم برآورده باز است. اگر برآمده ها روی نوار مغناطیسی به ترتیب شماره هایشان ذخیره شده باشند (یعنی برآمده ای اول در ابتدای نوار، برآمده دوم پس از آن، و ...). مقدار زمانی که طول می کشد تا بتوانیم برآمده ای شماره i را از روی نوار بخواهیم، مناسب با $\sum_{j=1}^{i-1} a_j = T$ است. شناسایی زمان خواهد داشت که برآمده از روی نوار به طور متوسط مناسب با $\frac{1}{n} \sum_{i=1}^n a_i = \bar{a}$ است. بدینسان زمان خواهد داشت که ترتیب برآمده ها روی نوار مغناطیسی را طوری تعیین کنیم که T حداقل شود.

- الف - با ازایه بیک مثال نایاب گشید که اگر برنامه ها را به ترتیب سعودی طول شان روی نوار دخیره کنند، ترتیب دخیره شدن برنامه ها لر روما نهیمه نیست.
- ب - با ازایه بیک مثال نایاب گشید که اگر برنامه ها به ترتیب نزولی طول شان روی نوار دخیره شوند، ترتیب دخیره شدن برنامه ها لر روما نهیمه نیست.
- ج - نایاب گشید که اگر برنامه ها را به ترتیب نزولی مقدار انحرافی نوار دخیره کنند، مقدار T مینیمم می شود، معنی آن ترتیب نهیمه است.

۹.۴ روش‌های جستجو

در روش‌های قبل روش‌های تفکیک و حل، بروبا و حریصانه که عمدتاً برای به دست آوردن راه حل‌های سریع و چندجمله‌ای برای مسائل استفاده می‌شوند، مورد بررسی قرار گرفت. ولی بینا کردی جن الکوریتم‌هایی کار ساده‌ای نیست و فهمی مسائل را نمی‌توان با روش‌های مذکور حل کرد.

اگر مسئله‌ای با هیچ یک از روش‌های قوی، قابل حل نبود، معکن است تها راه حل آن جستجوی وضایی حالت باشد؛ یعنی برای پیدا کردن حواب، گلبه‌ی حالت‌های مسئله را مورد بررسی قرار می‌دهم. این جستجو عموماً به یکی از روش‌های زیر احتمام می‌گیرد:

۹.۵ روش پس‌گرد

پس‌گرد^{۲۳} روشی است که گلبه‌ی قضایی حالت قابل تکمیل را با نظام خاصی انجام و جستجو می‌کند. به این صورت که در مرحله‌هایی که با جدین انتخاب روبرو می‌شود، یکی از انتخاب‌ها را بافرض درست بودن دنال می‌کند، اگر به جواب نرسید، با پس‌گرد (backtrack) انتخاب خود را عوض می‌کند و این کار را تا تمام شدن انتخاب‌های ادامه می‌دهد. این روش می‌تواند با مانع اولین جواب متفق شود و با این که جستجو را برای باقی گلبه‌ی جواب‌ها ادامه دهد.

الگوریتم‌های پس‌گرد عمولاً دارای هزینه‌ی نمایی می‌باشد.

مسئله‌ی هشت‌وزیر

```

for i1 := 1 to 8 do
  for i2:= 1 to 8 do
    for i3:= 1 to 8 do
      ...
      for i8:= 1 to 8 do
        try <- (i1,i2,...,i8)
        if solution (try) then write(try)
  
```

بردار k -promising

backtracking

```

procedure N-Queens (k, col, diag45, diag135)
try [1..k] is k-promising
  col= {try[i]: 1<= i <=k}
  diag45 = {try[i].i+1: 1<= i <=k}
  diag135 = {try[i].i-1: 1<= i <=k}
)
if k = N {an N-promising vector is a solution}
then write try
else { find (k+1)-promising extension}
  for j <= 1 to N do
    if (j not in col) and (j+k not in diag45)
      and (j+k not in diag135)
    then try[k+1] <- j
    N-Queens(k+1, col-{j}, diag45-{j+k}, diag135-{j+k})

```

برای $n = 12$ ، $n = 16$ و $n = 20$ های گشت و مود دارد. اولین جواب در 1546×44 ثانی حلقة بدست می‌آید. ولی درخت خالص در روش پیش‌گرد 856189 گره دارد که در 162 ثانی گره پک جواب بدست می‌آید.

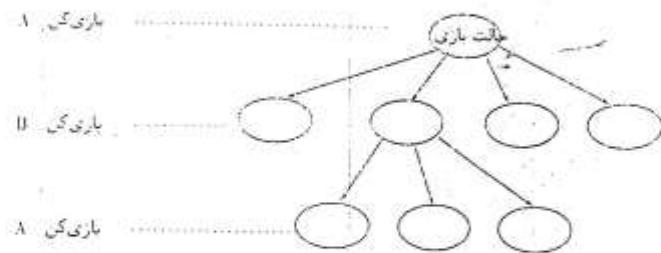
۲.۶.۴ درخت بازی

لکی از روش‌های حسروجوانی فضای مسئله استناده از درخت بازی^{۷۷} می‌باشد که جهت تعیین استراتژی مزد و احتمام سریع بازی مسکن نه هر مرحله به کار می‌رود. هر گره درخت سریع و خوبی از بازی دلالت می‌کند و رضیعت‌هایی که نا حرکت بعد قابل تولید هستند، به عنوان فرزندان این گره در نظر گرفته می‌شوند (ماسد شکل ۱.۴۶.۲).

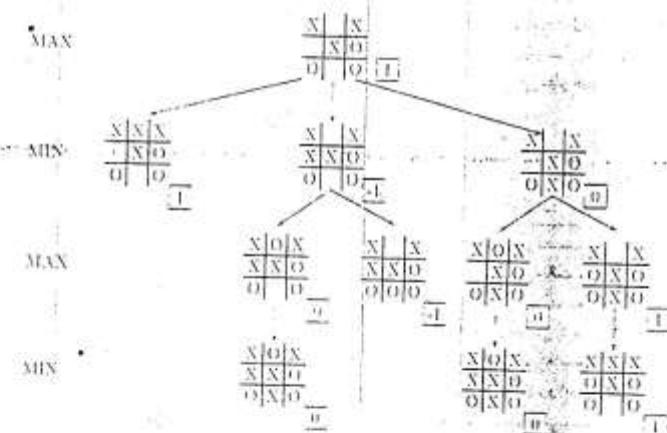
برگ‌ها حالتی بازی را نشان می‌دهند و هر مسیر را رسیدن به یک برگ، بیان گزینیک بازی است که بگ. بال این مسیر توسط نازیک اول انتساب شده و بال بعد را بازیکن درون انتساب گزیده است. به هر گره عددی سیست داده می‌شود که مسکنی مه رضیعت بازی دارد. هرای روش دهن مطلب به مثال و مرتوجه گذید:

در شکل ۱.۶.۲ از درخت بازی می‌توان به صورت سازمانی داده شده در شکل ۴۷.۴ در تدویم گزیری می‌رسیدن به درخت نهانی که هر گره دادن خروج مسایه به صورت دسال فرم می‌باشد. استناده مزد، در حالت نهانی (برگ‌ها) اگر X بزرگ‌تر ماسد عدد A ، اگر 0 (خریب) بزرگ‌تر ماسد عدد $-A$ و در غیر این صورت عدد هفقری به آن برگ نسبت داده می‌شود. در مطلع پدر اگر حرکت (سنجاب بال) از λ باشد پیشنهای عدد: فرزندان به این گره نسبت داده می‌شود و اگر حرکت از λ ماسد، کمینه‌ی آنها. مدن معنی که ماضی‌بینی را انتساب می‌کیم که بیشترین امسار را دارد و حرمت غم بهترین بازی خود را رایه می‌داند تا کمترین امتحار محبت ماند. از این رو این درخت را min-max درخت می‌گوییم.

۱.۴ روش های جستجو



شکل ۱.۴: درخت جستجو



شکل ۱.۴: درخت جستجو در بازی X-O

```

function Search(B: BoardType; Mode: ModeType): real;
var
  C: BoardType;
  Value: real;
begin
  if B is a leaf then
    return(Payoff(B))
  else
    begin
      if Mode = Max then
        Value := -1
      else
        Value := +1;
      for each child C of board B do
        if Mode = Max then
          Value := Max(Value, Search(C, Min))
        else
          Value := Min(Value, Search(C, Max));
      return(Value);
    end;
end;

```

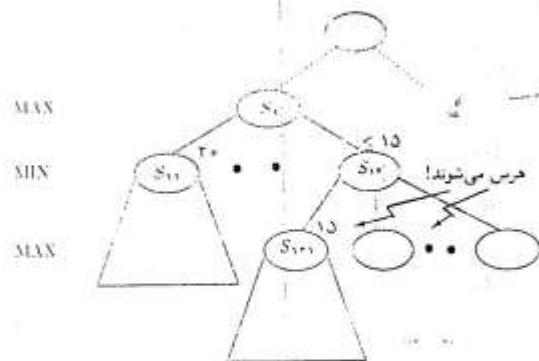
مسئله‌ای بروز نماید، فضای حالت را جست وجو می‌کند، اما در بعضی موارد به دلیل زیاد بودن حالت‌ها جنس امری امکان نداشته است. مثلاً در بازی شطرنج ساخت درخت نايسن به حالت‌هایی نهایی که بنوار به آن امصار بسته نمایند نمی‌تواند این را در جستجو کند. این امصار بارزه، ماهشه می‌شود و در سطح آخر با توجه به وضعیت مهرها در صفحه، بعدی را حدم زده، و به آن نسبت می‌دهم. هر قدر که این محدود قوی نزدیک تعداد سطوح اینجاد شده‌ی درخت می‌شود، به همین معنای بازی بعنوان انجام می‌گیرد.

۳.۷.۲ محدود کردن فضای جست وجو

بر مسئله‌ای بروز می‌شوند که بعای فضای حالت موردنیست وجو در این کنفرانس، اما با استفاده از روش‌های رسمندانه فضای جست وجو را کافی نمایند.

۱. هرسن کردن^{۲۰}

۲. اسماز و تجدید^{۲۱}



شکل ۷.۴: بازگردان S_{111} بی تاثیر بودن S_{111} منحص می گردد و نفعی فریدن آن درس می سود.

هرس گردن

گاهی اوقات جستجوی مرخی از رشته ها بین تابعی است و می توان از جستجوی آن صرف نظر کرد. بر شکل ۷.۴ جون S_{111} انتشاری معادل ۲۰ دارد. در نتیجه انتشار S_{111} هر دو رشته های S_{111} و S_{111} می بازگردانند که با انتشار ۱۵ متوجه می شوند که انتشار S_{111} کمتر از ۱۵ خواهد بود. و آن حاکم انتشار S_{111} می بازد. این انتشار از بازگردان نفعی فریدن آن صرف نظر می گشود و آن هارا می سامیم. بنابراین هر رشته های S_{111} می گشود. بر هر رشته های S_{111} این روش را هرس گوییم. شکل ۷.۶ نوبه ای روش را نشان می دهد.

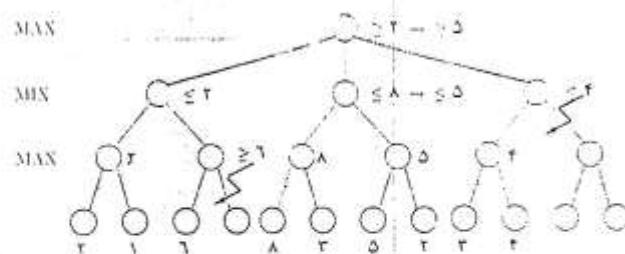
انتخاب و تحدید

بر اساس روش نیز مانند هرس گردن، از بازگردان مرخی از رشته ها خودداری می شود. به این صورت که ما بجزء و سفلی اطلاعات محلی موجود در بعضی از رشته ها، منحص می گردیم که این ساقه سهمی مطلوب حاصل می شود. بدین از بازگردان آن صرف نظر می شود. همچنین با توجه به نفس انتخاب، جستجو می گردد که اندام های از ساقه هایی غریب را موره حست و خود موارد داشته بازی روش ندی منحص به مدل را درج کند.

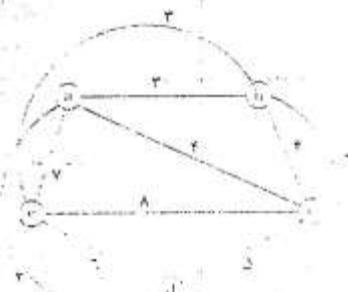
فروشنده ای دوره گرد

درین گنبد شهرها و راه هایی بین آن ها مطلق شکل ۷.۵ داده شده است. هدف بسیار گردان دوری با حداقل بورن ممکن است. که از کلیه ای راه ها عبور گند و از هر راه فقط یک پارسکرد.

حالت مسئله: تعادل از بمال ها انتخاب شده اند. منظمه این ها در مسیر مسیر و خوده دارند. همچنین تعادل ای راه ها خود شده اند. که بقیه بیان از آنها در مسیر مسیر استداده کرده. مثلاً گذشتی مسیر داشت که بیان های این راه را



شکل ۴۹.۴: نمونه‌ای از پرینگ (pruning)



سکن ۴-۲: میز نمایشگاه در مسئله‌ی فریموده‌ی نویز و تکرار.

۶.۴ دروس های حست و خر

سامان می ضود و بال ۲۰ در آن مسراها وجود ندارد. بک حالت از مسئله می باشد. این حالت را به حذف زیر مسماط می داشتم:

(۱) (۲) (۳)

تعیین حد: بر هر حالت برای هر رأس دو بال ما که ضرمن ورن، از بال های موجود، انتخاب می کنیم. دلیل این که هیچ مسیر های میتوانیم بینا شود که ورن آن کمتر از تصفیه مجموع ورن های این بال های باشد. این حد همان اطلاعات محلی در حالت محض می شود. مثلاً در حالتی که صفحه بالی حذف شده است، حد به قدر عرض محاسبه می شود:

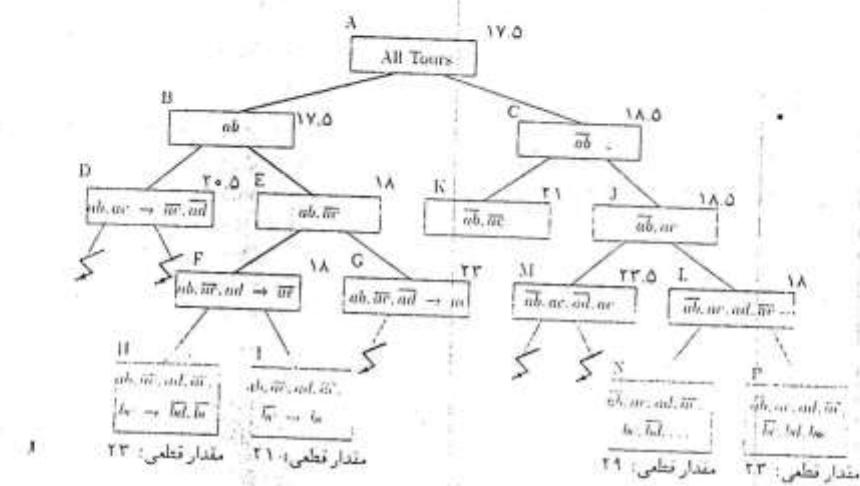
$$\begin{aligned} a &= 2 - 2 \\ b &= 2 + 2 \\ c &= 4 + 6 \\ d &= 2 + 5 \\ e &= 2 + 6 \end{aligned}$$

$$25 \div 4 = 17.5$$

لذا هیچ مسیری با وزن کمتر از ۱۷.۵ و در واقع کمتر از ۱۸ مداریم (جون زون ها عدد صحیح هستند).
تصمیم گیری برای بازگردان شاخه ها با توجه حد آن صورت می گیرد. همچنان در انتخاب بک شاخه برای دنبال کردن، اولویت را به شاخه ای می دهیم که حد کمتری دارد. برای مثال در شکل ۱۶.۴ ایندا مسیری که برزگرتر رسم شده، حست و جو می شود.

در شکل فوق پس از رسیدن به ۱ بازگشت و بازگشت به مرحله ۱۵ مرحله ۱۶ هرس می شود. زیرا مسیر راهی که از ۱۶ نولید می شود، برزگرتر با مساوی ۲۲ میشود. و همچنان در مرحله ۱۰ مسیر راهی که نولید می شود، برزگرتر با مساوی ۲۱ هم هرس می شود. در این مقطع بالاتر جون ۱۹ از ۱۹ برزگرتر است. لذا اعمال فوق را دوباره تکرار می کنیم و اتفاقاً با توجه به شکل مسیری به طول ۱۹ دست می آید که حوت مسئله ایست در حالی که D.G.M.V مازلشداد است.

نهنگی مهم در حل این گونه مسائل به دست آوردن بک حد ۱۹+۱۹+۱۹ صحیح می باشد.



شکل ۴.۵: درخت جستجوی مسئله فروشنده دور رنگ

