



مروزی بر ساختمان های داده

گردآورنده : امیر رضا توکلی

سخنی باشما

سلام ، خسته نباشید ؛

این خلاصه برای درس ساختمان های داده می باشد یعنی مباحث درس الگوریتم را پوشش نمی دهد که برای درس الگوریتم نیز جداگانه طراحی شده است که به زودی در اختیار شما قرار می گیرد ، در صورتی که سوالی داشتید می توانید از طریق راه های ارتباطی زیر با من در میان بگذارید.

و من الله توفيق ...

امیررضاتوکلی - تیر ۱۴۰۰

جهت هر گونه سوال از طریق شبکه ها اجتماعی با من در ارتباط باشید :

Phone / WhatsApp	+98 902 888 3854
Skype	Amirrezatavakoli3
Aparat	Amirrezatav
Telegram	Amirrezatav
Instagram	Amirrezatav5
Twitter	Amirrezatav5
LinkedIn	Amirrezatav
Facebook	Amirrezatav
Gmail	Amirrezatav2@gmail.com
Hotmail	Amirrezatavakoli3@hotmail.com
Rubika	Amirrezatav
GitHub	Amirrezatav
GitLab	Amirrezatav

در صورتی که مشکلی ، اشتباهی در حل سوالات یا درسنامه ، اشتباه تایپی و ... مشاهده کردید ، خوشحال می شوم از طریق لینک زیر یا اسکن QR Code با من در میان بگذارید.



<https://forms.gle/eJX1vi2AEjc4viVeA>

فهرست مطالب

Contents

۱	مروری بر ریاضی
۱	سیگما و دنباله ها
۲	لگاریتم
۲	خواص لگاریتم
۳	رشد توابع
۴	نماد های مجانبی
۶	زمان اجرای الگوریتم ها غیر بازگشتی
۱۰	زمان اجرای الگوریتم های بازگشتی
۱۰	جایگذاری از پایین(روشی ضعیف)
۱۰	جایگذاری از بالا
۱۱	معادله مشخصه
۱۴	درخت بازگشت
۱۶	قضیه master
۱۸	آرایه ها
۱۹	چند جمله ای
۲۲	ماتریس و ماتریس پراکنده(Sparse Matrix)
۲۶	لیست پیوندی
۲۸	مسئله جوزف (Josephus)
۲۹	لیست پیوندی - چند جمله ای
۳۲	جست و جو در آرایه (Liner Search)
۳۲	جست و جوی دودویی (برای آرایه های مرتب شده)
۳۴	جست و جو درون یاب (Interpolation search)
۳۶	جست و جو ترکیبی (Hybrid search)
۳۷	جست و جو در لیست پیوندی
۳۷	۱- جست و جو خطی
۳۷	۲- جست و جو دودویی
۳۸	مرتب سازی
۳۸	مرتب سازی انتخابی (Selection Sort)

۳۹	مرتب سازی حبابی (Bubble Sort)
۴۰	مرتب سازی درجی (Insertion Sort)
۴۱	مرتب سازی ادغامی (Merge Sort)
۴۳	Shell Sort
۴۴	درخت تصمیم
۴۵	پشته
۴۹	فرم های ریاضی
۵۲	ساخت چند پشته در یک آرایه
۵۳	The Maze Problem
۵۵	پیاده سازی مسئله Maze
۵۷	صف
۶۳	درهم سازی (Hashing)
۶۴	زنگیره سازی (Chaining)
۶۶	آدرس دهی باز (Open Addressing)
۷۰	تابع درهم ساز (Hash function)
۷۲	آنالیز استهلاکی (amortized analysis)
۷۲	تجمعی (aggregate)
۷۴	حسابرسی (accounting)
۷۶	درخت (Tree)
۷۹	درخت دودویی (Binary Tree)
۸۰	درخت دودویی پُر (Full Tree)
۸۲	درخت دودویی کامل
۸۳	نمایش و پیاده سازی درخت دودویی با آرایه
۸۶	نمایش و پیاده سازی درخت دودویی با لیست پیوندی
۸۶	پیمایش درخت
۹۲	پیاده سازی (کد) درخت با لیست پیوندی و پیمایش ها
۹۵	درخت عبارت ریاضی
۹۶	درخت عمومی (General Tree)
۹۷	نحوه نمایش یا پیاده سازی درخت عمومی
۹۷	تبدیل درخت عمومی به باینری

۹۸.....	پیمایش درخت عمومی.....
۹۸.....	جنگل.....
۹۸.....	پیمایش جنگل.....
۹۸.....	تبدیل جنگل به درخت دودویی
۹۹	درخت جست و جوی دودویی (BST)
۹۹	عملیات های پایه BST
۱۰۱.....	ساخت BST
۱۰۳.....	پیاده سازی BST
۱۰۷.....	هرم دودویی (Heap)
۱۰۷.....	هرم دودویی کمینه (Binary min Heap)
۱۰۷.....	هرم دودویی بیشینه (Binary min Heap)
۱۰۹	هر گره باید از فرزندانش بیشتر باشد پس max ریشه است.
۱۰۹	درج در هرم بیشینه
۱۱۱	عمل maxheapify
۱۱۱	ساخت هرم بیشینه
۱۱۳	حذف مکریم از هرم بیشینه (heap extractmax)
۱۱۳	روش مرتب سازی هرمی
۱۱۴	صف اولویت (Priority Queue)
۱۱۴	ساخت صف اولویت
۱۱۵	پیاده سازی هرم کمینه
۱۱۸	کوییز ها
۱۱۸	کوییز شماره ۱
۱۱۹.....	کوییز شماره ۲
۱۲۰	کوییز شماره ۳
۱۲۲	کوییز شماره ۳
۱۲۳	آزمون پایانی نیم سال دوم ۹۹
۱۲۶	خلاصه مرتبه های زمانی
۱۲۷.....	مجموعه های مجزا

مروری بر ریاضی

سیگما و دنباله ها

$$\sum_{i=1}^n i = \frac{n(n+1)}{2} \in \theta(n^2)$$

$$\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6} \approx \frac{n^3}{3} \in \theta(n^3) \quad (\text{تعداد مربع های شطرنج})$$

$$\sum_{i=1}^n i^3 \approx \frac{n^4}{4}$$

$$\sum_{i=1}^n \sqrt{i} \approx \frac{n^{2/3}}{\frac{2}{3}}$$

$$\left| \sum_{i=1}^n i^p = \frac{i^{p+1}}{p+1} \in \theta(n^{p+1}) \right.$$

$$\sum_{i=1}^n \frac{1}{i} = \ln(n) \in \theta(\log(n)) \quad (\text{سری هارمونیک یا همسازه})$$

$$\sum_{i=1}^{n=\infty} \left(\frac{1}{i}\right)^2 = 1$$

$$\sum_m^n a = (n - m + 1)a$$

$$\sum_{i=1}^n af(i) = a \sum_{i=1}^n f(i)$$

$$\sum_{i+1}^n af(i) + ag(i) = a(\sum_{i=1}^n f(i) + \sum_{i=1}^n g(i))$$

$$\sum_{i=1}^n f(i) - f(i+1) = f(1) - (n+1)$$

مثال : حاصل $\sum_{k=1}^n \frac{1}{k(k+1)}$ را بدست آورید. (پاسخ)

$$\text{sum}_{\text{تصاعد هندسی}} = \frac{a(1-q^n)}{1-q}$$

تصاعد هندسی با جملات نامحدود $f = a + aq + aq^2 + \dots + aq^n$ if $n \rightarrow \infty$ for $|q| < 1$ then $f = \frac{a}{1-q}$ and for $|q| > 1$ $f = \infty$

$$\text{sum}_{\text{حسابی}} = \frac{n(a_1 + a_n)}{2} = \frac{n(2a_1 + (n-1)d)}{2}$$

مثال : حاصل $\sum_{k=1}^n k(2^k)$ را بدست آورید. (پاسخ 2)

$$\begin{aligned}
 A &= 1 + x + x^2 + x^3 + \dots + x^n = \frac{1(1-x^{n+1})}{1-x} \xrightarrow[\substack{\cancel{x} \\ \cancel{1-x} \\ \cancel{(1-x^{n+1})}}]{\cancel{1-x}} A' = 1 + 2x + 3x^2 + \dots + nx^{n-1} = \frac{x^n(nx-n-1)+1}{(n-1)} \\
 &\xrightarrow{x \cdot x} A'x = x + 2x^2 + 3x^3 + \dots + nx^n = \frac{x^{n+1}(nx-n-1)+x}{(n-1)^2} \\
 &\text{if } x=2 \Rightarrow A'x = 2 + 2 \cdot 2^2 + 3 \cdot 2^3 + \dots + n \cdot 2^n = \frac{2^{n+1}(2n-n-1)+2}{(n-1)^2} = \underline{\underline{2^{\underline{n+1}}(n-1)^2}}
 \end{aligned}$$

مثال : حاصل $\sum_{k=0}^{\infty} \frac{k}{2^k}$ را بدست آورید. (پاسخ 2)

$$\sum_{n=1}^{\infty} \frac{n}{2^n} = \frac{(\frac{1}{2})^{n+1} - (\frac{1}{2}^{n-n-1}) + \frac{1}{2}}{(\frac{1}{2})^n} = \frac{(\frac{1}{2})^{n-1} - \frac{1}{2} - 1}{2} + 2 \xrightarrow{n \rightarrow \infty} 2$$

as for n = ∞

لگاریتم

$$f(x) = 2^x \rightarrow f^{-1}(x) = \log_2 x$$

$$\log_b a = c \rightarrow a = b^c$$

خواص لگاریتم

$$\log_b a + \log_b c = \log_b a \cdot c$$

$$\log_b a - \log_b c = \log_b a / c$$

$$\log_b a = \frac{1}{\log_a b}$$

$$\log_{b^m} a^n = \frac{n}{m} \log_b a$$

$$\log_b a = \frac{\log_c a}{\log_c b}$$

$$\log_b a * \log_c b * \log_d c = \log_d a$$

$$c^{\log_b a} = a^{\log_b c}$$

لگ استار : مقدار \log_n* برابر است با تعداد دفعاتی که باید از 2 لگاریتم بگیریم تا مقدار لگاریتم کمتر مساوی 1 باشد. (رشد بسیار بسیار پایینی دارد)

$$\log_2* = 1, \log_4* = 2, \log_{16}* = 3, \log_{65536}* = 4, \log_{2^{65536}}* = 5, \dots$$

در این درس تمامی لگاریتم ها بر پایه 2 در نظر گرفته شود.

رشد توابع

- بررسی رفتار تابع در بی نهایت (رشد برای مقادیر بزرگ معنی دارد و با نقدار فرق دارد)
- حد گیری :

$$k = \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \begin{cases} k = \infty, & f_{\text{رشد}} > g_{\text{رشد}} \\ 0 < k < \infty, & f_{\text{رشد}} = g_{\text{رشد}} \\ k = 0, & f_{\text{رشد}} < g_{\text{رشد}} \end{cases}$$

- ضریب ثابت تاثیری در رشد ندارد یعنی $kf(x)$ با $f(x)$ هم رشد است.
- لگاریتم ها با هر پایه ای هم رشد اند.
- نمایی ها از چند جمله ای ها رشد بیشتری دارند. $n^{200} > \text{رشد } 1.1^n$
- چند جمله ای ها از لگاریتمی ها رشد بیشتری دارند. $\text{رشد } (\log \text{چیز}) > \text{رشد } \log$
- اگر $f(x) > g(x)$ توان هرچه بیشتر \leftarrow رشد بیشتر
- اگر $f(x), g(x)$ باشند و هر دو صعودی آنگاه :

 - اگر $\text{رشد } (f(x)) > \text{رشد } (g(x))$ آنگاه رشد f بستر از g است (بلعکس درست نیست)
 - اگر $\text{رشد } (f(x)) < \text{رشد } (g(x))$ آنگاه رشد f بستر از g است (بلعکس درست نیست)
 - اگر $\text{رشد } (f(x)) = \text{رشد } (g(x))$ آنگاه نمی توان نظری داد
 - در نتیجه نکته قبل برای مقایسه رشد میتواند از دو تابع لوگاریتم گرفت.
 - رشد n^n از رشد $n!$ بیشتر است، ولی $\text{رشد } (\log n)^{\log n} = \text{رشد } (\log n)^{\log n}$. (اثبات با تقریب استرلینگ)
 - $[n] = [n]$ برابر است با اولین عدد کوچکتر یا مساوی ولی $[n]$ برابر است با اولین عدد بزرگتر یا مساوی n .
 - کف و سقف گاهی اوقات مهم نیست ولی گاهی اوقات اهمیت پیدا میکند
 - در جمع و تفریق مقدار رشد \max همان رشد چند جمله ای است.

مثال : ترتیب رشد $2^{\log n}, (\log n)^n, n^{\log \log n}, (\log n)^{\log n}, 2^n, n^2$ را بدست آورید.

$$(\log n)^n > 2^n > 2^{\log n} > n^{\log \log n} = (\log n)^{\log n}$$

$$\log(2^{\log n}) = \log n \quad \Rightarrow \quad 2^{\log n} < (\log n)^n \quad \textcircled{1} \quad n^2 < 2^{\log n} \quad \textcircled{2} \quad 2^n < (\log n)^{\log n} \quad \textcircled{3}$$

$$\log((\log n)^{\log n}) = \log \log n \cdot \log n \quad \left\{ \begin{array}{l} \log \log n = \log n \\ \log n = (\log n)^1 \end{array} \right. \quad \textcircled{4}$$

$$= 2^{\log n} > 2^{\log \log n} > n^{\log \log n} = (\log n)^{\log n}$$

| مثال : ترتیب رشد $(|loglogn|! < n^2 < |logn|! < 2^n, |logn|!, |loglogn|!, 2^n, n^2)$ را بدست آورید. (پاسخ :

نماد های مجانبی

➤ آنالیز رفتار الگوریتم به ازای ورودی های بزرگ.

➤ انواع نماد ها $\Theta, o, \Omega, \omega, \theta$

➤ نماد O

$$f(x) \in O(g(x)) \text{ or } f(x) = O(g(x)) \quad \bullet$$

مفهوم : رشد f از رشد g کمتر یا مساوی است.

$f(n) \leq c \cdot g(n)$ می خوانیم : تابع f از مرتبه $O(g)$ است هرگاه عدد c پیدا شود که نا مساوی

بعد از یک نقطه مانند $n_0 > n$ برقرار باشد. به ازای یک c درست باشد کافی است.

➤ نماد Ω

$$f(x) \in \Omega(g(x)) \text{ or } f(x) = \Omega(g(x)) \quad \bullet$$

مفهوم : رشد f از رشد g بیشتر یا مساوی است.

$f(n) \geq c \cdot g(n)$ می خوانیم : تابع f از مرتبه $\Omega(g)$ است هرگاه عدد c پیدا شود که نا مساوی

بعد از یک نقطه مانند $n_0 > n$ برقرار باشد. به ازای یک c درست باشد کافی است.

➤ نماد o

$$f(x) \in o(g(x)) \text{ or } f(x) = o(g(x)) \quad \bullet$$

مفهوم : رشد f از رشد g کمتر است.

$f(n) < c \cdot g(n)$ می خوانیم : تابع f از مرتبه $o(g)$ است هرگاه عدد c پیدا شود که نا مساوی

بعد از یک نقطه مانند $n_0 > n$ برقرار باشد. به ازای یک c درست باشد کافی است.

➤ نماد ω

$$f(x) \in \omega(g(x)) \text{ or } f(x) = \omega(g(x)) \quad \bullet$$

مفهوم : رشد f از رشد g بیشتر است.

$f(n) > c \cdot g(n)$ می خوانیم : تابع f از مرتبه $\omega(g)$ است هرگاه عدد c پیدا شود که نا مساوی

بعد از یک نقطه مانند $n_0 > n$ برقرار باشد. به ازای یک c درست باشد کافی است.

➤ نماد θ

$$f(x) \in \theta(g(x)) \text{ or } f(x) = \theta(g(x)) \quad \bullet$$

$f(x) \in \theta(g(x)) \Leftrightarrow f(x) \in \Omega(g(x)) \cap f(x) \in O(g(x))$ و f هم رشد اند.

مفهوم : تابع f از مرتبه $\theta(g)$ است هرگاه عدد $c_1, c_2 > 0$ پیدا شود که نا مساوی

$$c_1g(x) < f(x) < c_2g(x) \quad \text{بعد از یک نقطه مانند } n_0 > n \text{ برقرار باشد}$$

➤ بنابراین داریم :

رشد f کمتر مساوی g $f(x) \in O(g(x))$ -۱

رشد f بیشتر مساوی g $f(x) \in \Omega(g(x))$ -۲

رشد f کمتر از g $f(x) \in o(g(x))$ -۳

رشد f بیشتر از g $f(x) \in \omega(g(x))$ -۴

رشد f هم رشد g $f(x) \in \theta(g(x))$ -۵

نمی توان ادعا کرد که همواره $f(x) \in O(g(x))$ یا $f(x) \in \Omega(g(x))$ یا $f(x) \in \theta(g(x))$

$$n \neq O(n^{\sin(n)+1}) \text{ and } n \neq \Omega(n^{\sin(n)+1})$$

هرجا تناوب دیدیم شرایط بالا برقرار است.

خواص نماد های مجانبی :

-۱ بازتابی (reflection) : هر عضو با خودش رابطه دارد :

$$(f \in O(f), f \in \Omega(f), f \in \theta(f)) \quad \blacksquare$$

-۲ تقارن (symmetric) :

$$f = \theta(g) \Leftrightarrow g = \theta(f) \quad \blacksquare$$

با جایه جایی طرفین علامت ها تغییر میکند. Transpose symmetric -۳

$$f \in \Omega(g) \Leftrightarrow g \in O(f) \quad \blacksquare$$

$$f \in \omega(g) \Leftrightarrow g \in o(f) \quad \blacksquare$$

-۴ تعددی (transitive) :

برای همه نماد ها برقرار است.

$$f \in \theta(h) \Leftrightarrow \begin{cases} f \in \theta(g) \\ g \in \theta(h) \end{cases} \quad \blacksquare$$

ماکسیمم گیری :

-۱ زمانی درست است که تعداد جملات ثابت باشد.

مثال : نشان دهید اگر $k = \theta\left(\frac{n}{\ln(n)}\right)$ آنگاه $k \cdot \ln(k) = \theta(n)$

$$K \ln(K) = \theta(n) \Rightarrow n = \theta(K \ln(K)) \xrightarrow{\ln} \ln n = \theta(\ln K + \ln \ln K) = \theta(\ln K)$$

$$\begin{aligned} \ln n &= \theta(K \cdot \ln K) \\ \ln n &= \theta(\ln K) \quad \therefore \quad \frac{n}{\ln n} = \theta(K) \Rightarrow K \leq \theta\left(\frac{n}{\ln(n)}\right) \end{aligned}$$

مثال : آیا $O(f) - \Omega(f) == o(f)$ لزوماً خیر

$$\text{if } f \in \begin{cases} n & \text{odd} \\ n^2 & \text{even} \end{cases} \Rightarrow \begin{cases} f \in O(n) \\ f \notin \Omega(n) \end{cases} \Rightarrow f \in O(n) - \Omega(n) \quad \times$$

$\begin{matrix} f \in O(n) \\ f \notin \Omega(n) \end{matrix} \Rightarrow f \notin o(f)$

➤ زمان اجرای الگوریتم به موارد زیر وابسته است :

- ۱- پیچیدگی الگوریتم
- ۲- سخت افزار
- ۳- کامپایلر (زمان اجرای هر دستور)
- ۴- اندازه ورودی

➤ گاهی نمیتوان زمان دقیق اجرای الگوریتم را محاسبه کرد بنابراین در حالت ها beset, average, worst می سنجیم.

زمان اجرای الگوریتم ها غیر بازگشتی

• برابر است با زمان اجرا تمام خطوط.

مثال : همه دستورات جمعاً چند با اجرا می شود؟

$$\begin{aligned} 1. \text{for } (\text{int } i = 1; i \leq n; i++) &\rightarrow n+1 \\ 2. \quad \quad \quad \text{cout} \ll "*" ; &\rightarrow \text{بار } n \quad \left\{ \Rightarrow 2n+1 \in \Theta(n) \right. \end{aligned}$$

مثال : همه دستورات جمعاً چند با اجرا می شود؟

$$\begin{aligned} 1. \text{for } (\text{int } i = 1; i \leq n; i++) &\rightarrow n+1 \\ 2. \quad \quad \quad \text{for } (\text{int } j = 1; j \leq n; j++) &\rightarrow n(n+1) \quad \left\{ + = 2n^2 + 2n + 1 = \Theta(n^2) \right. \\ 3. \quad \quad \quad \text{cout} \ll "*" ; &\rightarrow n \ll n \end{aligned}$$

مثال : همه دستورات جمعاً چند با اجرا می شود؟

$$\begin{aligned} 1. \text{for } (\text{int } i = 1; i \leq n; i++) &\rightarrow n+1 \\ 2. \quad \quad \quad \text{for } (\text{int } j = 1; j \leq n; j++) &\rightarrow n(n+1) \\ 3. \quad \quad \quad \text{for } (\text{int } k = 1; k \leq n; k++) &\rightarrow n^2(n+1) \\ 4. \quad \quad \quad \text{cout} \ll "*" ; &\rightarrow n^3 \quad \left\{ + = 2n^3 + 2n^2 + 2n + 1 = \Theta(n^3) \right. \end{aligned}$$

مثال : مرتبه اجرای برنامه زیر را بدست آورید ؟

$$\begin{aligned} 1. \text{for } (\text{int } i = 1; i \leq n; i++) &\rightarrow n+1 \\ 2. \quad \quad \quad \text{for } (\text{int } j = 1; j \leq i; j++) &\rightarrow 2+3+\dots+n+1 \\ 3. \quad \quad \quad x = x + 1; &\rightarrow 1+2+\dots+n \quad \rightarrow \sum_{i=0}^n \sum_{j=0}^i i = 0+1+2+\dots+n = \frac{n(n+1)}{2} \quad \Rightarrow \Theta(n^2) \end{aligned}$$

مثال : تعداد دفعات اجرای cout را بدست آورید ؟

```
for (int i = 1; i <= n; i++)
    for (int j = 1; j <= i; j++)
        for (int k = 1; k <= j; k++)
            cout << "*";
```

روش اول :

for (int i = 1; i <= n; i++)
 for (int j = 1; j <= i; j++)
 for (int k = 1; k <= j; k++)
 cout << "*";

$$\sum_{i=1}^n \sum_{j=1}^i \sum_{k=1}^j 1 = \sum_{i=1}^n \frac{i(i+1)}{2} = \frac{1}{2} \sum_{i=1}^n i^2 + i = \Theta(n^3)$$

روش دوم :

هر بار اجرا cout شامل دسته ای از (i, j, k) است به صورتی که $1 \leq k \leq j \leq i \leq n$ می باشد یعنی ترکیب باتکرار سه شی از n شی

$${n+3-1 \choose 3} = \frac{n(n+1)(n+2)}{6}$$

روش سوم :

for (int i = 1; i <= n; i++)
 for (int j = 1; j <= i; j++)
 for (int k = 1; k <= j; k++)
 cout << "*";

مثال : تعداد دفعات اجرای cout را بدست آورید ؟

```
for (int i = 1; i <= n; i++)  

  for (int j = 1; j <= n; j+=i)  

    cout << "*";
```

$$\begin{aligned} \rightarrow i = 1 &\rightarrow 1 \cdot n \\ i = 2 &\rightarrow 1 \cdot \frac{n}{2} \\ i = 3 &\rightarrow 1 \cdot \frac{n}{3} \\ &\vdots \end{aligned} \Rightarrow n + \frac{n}{2} + \frac{n}{3} + \dots = n \sum_{i=1}^n \frac{1}{i} \in \Theta(\ln n)$$

مثال : مرتبه اجرای برنامه را بدست آورید ؟

```
for (int i = 1; i <= n; i++)  

  for (int j = 1; j <= n; j++)  

  {  

    cout << "*";  

    n--;  

  }
```

$$\begin{cases} \rightarrow i = 1 \rightarrow \frac{n}{2} \cdot n \\ \rightarrow i = 2 \rightarrow \frac{n}{4} \cdot n \Rightarrow \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \dots \in \Theta(n) \\ \rightarrow i = 3 \rightarrow \frac{n}{8} \cdot n \\ \vdots \end{cases}$$

مثال : مرتبه اجرای برنامه را بدست آورید ؟

```
for (int i = 1; i <= n; i++)
{
    for (int j = 1; j <= n; j++)
        cout << "*";
    n--;
}
```

$$\begin{aligned}
 i &\leq 1 \rightarrow n/2 \\
 i &\leq 2 \rightarrow n-1/2 \\
 i &\leq 3 \rightarrow n-3/2 \\
 &\vdots \\
 i &\leq \frac{n}{2} \rightarrow \frac{n}{2}/2
 \end{aligned}
 \quad \text{بر علیش می نماییم}$$

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

مثال : مرتبه اجرای برنامه را بدست آورید ؟

```
for (int i = 1; i <= n; i*=2) \rightarrow i \leq 1, 2, 4, 8, \dots 2^k = n \rightarrow k \leq \log n = \Theta(\log n)
```

مثال : مرتبه اجرای برنامه را بدست آورید ؟

```
for (int i = 1; i <= n; i*=2) \leftarrow K+1/2 \quad K = \log n
    for (int j = 1; j <= n; j++) \leftarrow K+1(n)
        cout << "*"; \leftarrow K

```

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

مثال : مرتبه اجرای برنامه را بدست آورید ؟

```
for (int i = 1; i <= n; i*=2) \leftarrow K+1/2 \quad K = \log n
```

$$\begin{aligned}
 &\text{for (int j = 1; j <= i; j++)} \\
 &\quad \text{cout} << "*"; \rightarrow i \leq 1 \rightarrow 1/2 \\
 &\quad \rightarrow i \leq 2 \rightarrow 1/2 \quad \leftarrow 1+2+4+8+\dots+2^K, 2^{K+1} \leq n-1 \\
 &\quad \rightarrow i \leq 4 \rightarrow 1/4 \\
 &\quad \rightarrow i \leq K \rightarrow 1/2^K
 \end{aligned}$$

$$= \Theta(n)$$

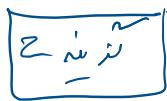
مثال : مرتبه اجرای برنامه را بدست آورید ؟

```
for (int i = 1; i <= n; i++)
    for (int j = 1; j <= i; j*=2)
        cout << "*";
```

$$\begin{aligned}
 &i | 1 2 3 4 \dots 8 \\
 &| 1 3 \\
 &| 1 2 2 3 \dots 4 \leq 1+2+2+3+3+3+4+\dots \\
 &= \sum_{i=1}^n \log i \geq \log 1 + \log 2 + \log 4 = \log(1 \times 2 \times \dots \times n) = \log n! \leq n \log n
 \end{aligned}$$

مثال : cout دقيقاً چند بار تکرار می شود ؟

```
i = 1;
while (i<n)
{
    cout<< "*";
    i*=2;
}
```



مثال : cout دقيقاً چند بار تکرار می شود ؟

```
i = n;
while (i>1)
{
    cout<<"*";
    i = (int)i/2;
}
```

$n \leq 2 \rightarrow 1$
 $n \leq 3 \rightarrow 1$
 $n \leq 4 \rightarrow 2 \Rightarrow \underbrace{1}_{\text{لزمه}}$
 $n \leq 7 \rightarrow 2$
 $n \leq 8 \rightarrow 3$

مثال : cout دقيقاً چند بار تکرار می شود ؟

```
i = 2;
while (i<=n)
{
    cout<<"*";
    i *= i;
}
```

$i = 2, 4, 16, 256, \dots 2^k = n \Rightarrow 2^k \leq n \Rightarrow k \leq \log n \Rightarrow \boxed{k = \lceil \log n \rceil}$

مثال : cout دقيقاً چند بار تکرار می شود ؟

```
i = 2;
while (i<=n)
{
    cout<<"*";
    i = i*i*i;
}
```

$i = 2, 8, 64, \dots 2^k = n \Rightarrow 3^k \leq n \Rightarrow k \leq \log_3 n \Rightarrow \boxed{k = \lceil \log_3 n \rceil}$

مثال : فرض کنید n نفر با قدرت های متفاوت به صورت تصادفی یکی وارد کلاس نقاشی می شوند ، کلاس یک مسئول دارد که با ورود هر شخص قد آن را اندازه می گیرد و اگر در بین تمام افرادی که وارد کلاس شده اند فرد قد بلند ترین باشد قدش را یادداشت میکند ، این مسئول به طور متوسط چند بار عمل یاد داشت را انجام می دهد.

در مثال بالا می خواهیم max پیدا کنید که کد آن به صورت زیر می باشد :

```
int A[1...n];
int max = 0;
for (int i = 0; i < n; i++)
{
    if(A[i] > max)
        max = A[i];
}
```

حالت i حالت \neg بر عمل یاد داشت اخیر هست .
 حالت n بر عمل یاد داشت لبیم هست
 حالت $\text{حصیر} :$

$$\text{avge} = \frac{1}{n} + \frac{1}{2} + \dots + \frac{1}{n} \leq \lceil \log n \rceil$$

حالت متوسط یا امید ریاضی $(\text{زمان هر حالت} * \text{احتمال هر حالت})$ به ازای هر حالت $\sum = (\text{امید ریاضی})$

زمان اجرای الگوریتم های بازگشتی

- » رابطه بازگشتی : در این روابط جمله a_n بر حسب جملات قبلی بیان می شود.
- » حل روابط بازگشتی روش ها مختلفی دارد که چند مورد بررسی شده است.

جایگذاری از پایین (روشی ضعیف)

مثال : رابطه بازگشتی 1 با فرض $a_0 = 0$ $a_n = 2a_{n-1} + 1$

$$a_0 = 0 \quad a_1 = 2a_0 + 1 = 1 \quad a_2 = 2a_1 + 1 = 3 \quad a_3 = 7, \quad a_4 = 15 \\ \dots, 1, 3, 7, 15, \dots \Rightarrow a_n = 2^n - 1$$

جایگذاری از بالا

مثال : رابطه بازگشتی 1 با فرض $a_0 = 0$ $a_n = 2a_{n-1} + 1$

$$a_n = 2a_{n-1} + 1 = 2(2a_{n-2} + 1) + 1 = 4a_{n-2} + 3 = 4(2a_{n-3} + 1) + 1 + 2 \\ = 8a_{n-3} + 1 + 2 + 4 + \dots = \underbrace{2^na_0 + 1 + 2 + 4 + \dots + 2^{n-1}}_0 = 2^{n-1} - 1$$

مثال : رابطه بازگشتی 1 با فرض $a_0 = 0$ $a_n = a_{n-1} + n$

$$a_n = a_{n-1} + n = a_{n-2} + n - 1 + n = a_{n-3} + n - 2 + n - 1 + n = \dots = a_0 + 1 + 2 + \dots + n = \frac{n(n+1)}{2}$$

مثال : رابطه بازگشتی 1 با فرض $a_0 = 0$ $a_n = a_{n-1} + \frac{1}{n}$

$$a_n = a_{n-1} + \frac{1}{n} = a_{n-2} + \frac{1}{n-1} + \frac{1}{n} + \dots + a_0 + 1 + \frac{1}{2} + \dots + \frac{1}{n} \approx \ln(n) \leq \Theta(\log n)$$

مثال : رابطه بازگشتی 1 با فرض $T_{(1)} = 0$ $T_{(n)} = T_{(n-1)} + \log n$

$$T_{(n)} \leq T_{(n-1)} + \log n = T_{(n-2)} + \log(n-1) + \log n = \dots = T_{(1)} + \log(1) + \log(2) + \log(3) + \dots + \log n = \log n! = \Theta(n \log n)$$

مثال : رابطه بازگشتی 1 با فرض $T_{(1)} = 0$ $T_{(n)} = 2T_{(\frac{n}{2})} + n - 1$ و n توانی از ۲ حل کنید.

$$T(n) \leq 2T\left(\frac{n}{2}\right) + n - 1 = 2\left(2T\left(\frac{n}{4}\right) + \frac{n}{2} - 1\right) + n - 1 = 2^2T\left(\frac{n}{4}\right) + n - 2 + n - 1 = 2^3T\left(\frac{n}{8}\right) + n - 2^2 + n - 2 + n - 1 \\ = \dots = 2^{\frac{n}{2}}T\left(\frac{n}{2^{\frac{n}{2}}}\right) + n - 2 + n - 2 + \dots + n - 2 + n - 1 \leq 2^{\frac{n}{2}} + 2^{\frac{n}{2}-1} + \dots + 1 \leq 1 + 2 + 4 + \dots + 2^{\frac{n}{2}} = 2^{\frac{n}{2}+1} - 1 \leq \boxed{2^{\frac{n}{2}+1} \leq n \log n + n + 1}$$

معادله مشخصه

۱. همگن:

فرض کنید رابطه بازگشتی به صورت زیر است:

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + c_3 a_{n-3} + \dots + c_k a_{n-k}$$

برای این رابطه بازگشتی معادله مشخصه به صورت:

$$x^k = c_1 x^{k-1} + c_2 x^{k-2} + c_3 x^{k-3} + \dots + c_k$$

بعد از بدست آوردن معادله آن را حل میکنیم تا ریشه ها بدست آید سپس به بررسی ریشه ها می پردازیم:

- اگر دو ریشه متمایز x_1, x_2 داشته باشد $a_n = \alpha_1(x_1)^n + \alpha_2(x_2)^n$ که α_1, α_2 با جایگذاری شرایط اولیه بدست می آید.
- اگر سه ریشه متمایز x_1, x_2, x_3 داشته باشد $a_n = \alpha_1(x_1)^n + \alpha_2(x_2)^n + \alpha_3(x_3)^n$ که $\alpha_1, \alpha_2, \alpha_3$ با جایگذاری شرایط اولیه بدست می آید.
- اگر دو ریشه برابر $x_1 = x_2$ باشد $a_n = (\alpha_1 + n \cdot \alpha_2)(x_1)^n$ که α_1, α_2 با جایگذاری شرایط اولیه بدست می آید.
- اگر سه ریشه برابر $x_1 = x_2 = x_3$ داشته باشد $a_n = (\alpha_1 + n \cdot \alpha_2 + n^2 \cdot \alpha_3)(x_1)^n$ که $\alpha_1, \alpha_2, \alpha_3$ با جایگذاری شرایط اولیه بدست می آید.
- اگر دو ریشه برابر $x_1 = x_2$ و یک ریشه متمایز x_3 داشته باشد $a_n = (\alpha_1 + n \cdot \alpha_2)(x_1)^n + \alpha_3(x_3)^n$ که $\alpha_1, \alpha_2, \alpha_3$ با جایگذاری شرایط اولیه بدست می آید.
- برای سایر حالت ها نیز به روش های مشابه قابل پیگیری است.

مثال: رابطه بازگشتی $a_n = a_{n-1} + a_{n-2}$ با فرض $a_0 = 0, a_1 = 1$ (دباله فیبوناتچی):

$$\begin{aligned} a_n &= a_{n-1} + a_{n-2} \Rightarrow x^2 = x + 1 \Rightarrow x^2 - x - 1 = 0 \Rightarrow (x - \frac{1}{2})^2 - \frac{5}{4} = 0 \\ &\Rightarrow x - \frac{1}{2} = \pm \frac{\sqrt{5}}{2} \Rightarrow x_1 = \frac{1 + \sqrt{5}}{2}, x_2 = \frac{1 - \sqrt{5}}{2} \\ a_n &= \alpha_1 x_1^n + \alpha_2 x_2^n \Rightarrow \begin{cases} a_0 = 0 \Rightarrow \alpha_1 + \alpha_2 = 0 \\ a_1 = 1 \Rightarrow \alpha_1 x_1 + \alpha_2 x_2 = 1 \end{cases} \Rightarrow \begin{cases} \alpha_1 = -\alpha_2 \\ \alpha_1 = \frac{1}{x_1} = \frac{1}{1 + \sqrt{5}/2} = \frac{2}{1 + \sqrt{5}} \\ \alpha_2 = -\alpha_1 = \frac{2}{1 - \sqrt{5}} \end{cases} \\ \Rightarrow \alpha_2 &= \frac{1}{\frac{1 - \sqrt{5}}{2}} = -\frac{1}{\sqrt{5}}, \alpha_1 = \frac{1}{\sqrt{5}} \end{aligned}$$

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

مثال : رابطه بازنگشتی $a_n = 4a_{n-1} - 4a_{n-2}$ با فرض $a_0 = 1, a_1 = 2$

$$x^2 - 4x + 4 \Rightarrow x^2 - 4x + 4 = (x-2)^2 = 0 \Rightarrow x_1 = x_2 = 2$$

$$a_n = (\alpha_1 + n\alpha_2)(2)^n \quad \left[\begin{array}{l} \alpha_1 = 2 \\ \alpha_2 = 1 \end{array} \right]$$

$$\alpha_1 = 1 + \alpha_2 \cdot 2 \Rightarrow 2 = 1 + \alpha_2 \Rightarrow \alpha_2 = 1$$

$$a_n = (2)^n$$

مثال : رابطه بازنگشتی $2a_n - 7a_{n-1} + 7a_{n-2} - 2a_{n-3} = 0$ با فرض $a_0 = 1, a_1 = 1, a_2 = 2$

$$\rightarrow 2x^3 - 7x^2 + 7x - 2 = 0 \quad \left[\begin{array}{l} x=1 \\ x=2 \\ x=-1 \end{array} \right] \rightarrow 2-7+7-2=0$$

$$\rightarrow 2x^3 - 7x^2 + 7x - 2 \quad \left[\begin{array}{l} x=1 \\ x=2 \\ x=-1 \end{array} \right]$$

$$\frac{-2x^3 + 2x^2}{-5x^2 + 7x} \quad \rightarrow (2x^2 - 5x + 2)(x-1) = 0$$

$$\begin{array}{r} -5x^2 + 7x \\ + 5x^2 - 5x \\ \hline 2x - 2 \\ -2x + 2 \\ \hline 0 \end{array}$$

$$\Delta = 25 - 16 = 9 \quad \left[\begin{array}{l} x_1 = 1 \\ x_2 = \frac{5-3}{4} = \frac{1}{2} \\ x_3 = \frac{5+3}{4} = 2 \end{array} \right]$$

$$\Rightarrow a_n = \alpha_1(1)^n + \alpha_2\left(\frac{1}{2}\right)^n + \alpha_3(2)^n \quad \left[\begin{array}{l} \text{بازنگشتی} \\ \text{اولین} \end{array} \right] \quad \left[\begin{array}{l} \alpha_1 = 1, \alpha_2 = \frac{1}{3}, \alpha_3 = \frac{2}{3} \end{array} \right]$$

$$\Rightarrow a_n = 1 + \frac{1}{3}\left(\frac{1}{2}\right)^n + \frac{2}{3}(2)^n = \Theta(2^n)$$

۲. ناهمگن :

فرض کنید رابطه بازنگشتی به صورت زیر است :

$$a_n = c_1a_{n-1} + c_2a_{n-2} + c_3a_{n-3} + \dots + c_ka_{n-k} + f_{(n)}$$

که $f_{(n)}$ به صورت $p(n)$ چند جمله ای درجه d است .

برای این رابطه بازنگشتی معادله مشخصه به صورت :

$$(x^k - c_1x^{k-1} - c_2x^{k-2} - c_3x^{k-3} - \dots - c_k)(x - b)^{d+1} = 0$$

• برای ریشه ها شرایط قبل برقرار است .

مثال : رابطه بازگشتی $a_n = a_{n-1} + n$ با فرض $a_0 = 0$

$$a_n = a_{n-1} + n \rightarrow (1)^n (n) \leq 1$$

$$(n-1)(n-1)^2 = 0 \Rightarrow n_1 < n_2 < n_3 \leq 1$$

$$a_n = (\alpha_1 + n\alpha_2 + n^2\alpha_3) (1)^n \rightarrow a_0 = \alpha_1 \leq 0$$

$$\alpha_1 \leq n_2 + n_3 \leq 1$$

$$\alpha_2 \leq 2\alpha_2 + 4\alpha_3 \leq 3 \quad \left\{ \rightarrow \alpha_2 \leq \alpha_3 \leq \frac{1}{2} \right.$$

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

$$\alpha_0 \leq 0$$

$$\alpha_1 \leq \alpha_0 + 1 \leq 1$$

$$\alpha_2 \leq \alpha_1 + 1 \leq 3$$

مثال : شکل جواب رابطه بازگشتی $a_{n+1} = 4a_{n-1} + 2^{n+4} + n * 2^n + 5n + 6$ بدست آورید

$$(n^2 - 4)(n-2)^2(n-1)^2$$

$$\overbrace{2^{(1b+n)} + (1)^{(5n+6)}}^{\uparrow d=1 \quad \uparrow d=1}$$

$$n \leq 2, n \leq 2, n \leq 1$$

$$\Rightarrow a_n = (\alpha_1 + n\alpha_2)(1)^n + \alpha_3(n-2)^n + (\alpha_4 + n\alpha_5 + n^2\alpha_6)2^n$$

مثال : رابطه بازگشتی $a_n = a_{n-1} \cdot a_{n-2}^2$ بدست آورید

$$a_n = a_{n-1} \cdot a_{n-2}^2 \xrightarrow{\log} \log a_n = \log a_{n-1} + 2 \log a_{n-2}$$

$$\log a_n = b_n \quad *$$

$$\Rightarrow b_n = b_{n-1} + 2b_{n-2} \rightarrow n^2 - n - 2 \leq 0 \rightarrow (n-\frac{1}{2})^2 - \frac{9}{4} \leq 0 \Rightarrow n-\frac{1}{2} \leq \pm \frac{3}{2}$$

$$\Rightarrow n_1 \leq \frac{-1+3}{2} \leq 1, n_2 \leq \frac{-1-3}{2} \leq -1 \Rightarrow b_n = \alpha_1(-1)^n + \alpha_2(-1)^n \Rightarrow \alpha_1 \leq \frac{1}{3}, \alpha_2 \leq -\frac{1}{3}$$

$$\log a_n = b_n \rightarrow \log a_0 = b_0 \leq 0 \quad *$$

$$\log a_1 = b_1 \leq 1 \quad *$$

$$\Rightarrow b_n = \frac{1}{3} - \frac{(-1)^n}{3}$$

$$\Rightarrow a_n = \frac{b_n}{2} = \frac{\frac{2^n}{3} - \frac{(-1)^n}{3}}{2}$$

مثال : مرتبه زمانی n را بدست آورید.

$$\begin{aligned} T(n)^2 &= \frac{1}{2}T(n-1)^2 + \frac{1}{2}T(n-2)^2 + n \Rightarrow a_n = \frac{a_{n-1}}{2} + \frac{a_{n-2}}{2} + n \\ &\rightarrow (x^2 - n - 1)(x - 1)^2 = 0 \rightarrow x_1 = n_2, x_3 = 1, x_4 = \frac{1}{2} \\ (x_1 + x_2 + n x_3) &(1) + x_4 (-\frac{1}{2}) \Rightarrow a_n = \Theta(n^2) \Rightarrow T(n) \leq \sqrt{a_n} = \Theta(n) \end{aligned}$$

مثال : مطلوب است $n \cdot T(n) = 3(n-1) \cdot T(n-1) - 2(n-2)T(n-2)$ را بدست آورید.

$$n \cdot T(n) \leq a_n \Rightarrow a_n \leq 3a_{n-1} - 2a_{n-2} \Rightarrow x^2 - 3x + 2 \leq 0 \quad \dots \quad \text{روش جای قابل}$$

مثال : مطلوب است $a_0 = 1, a_1 = 2$. $n \cdot T(n) = 3n \cdot T(n-1) - 2n(n-1)T(n-2)$ را بدست آورید.

$$\begin{aligned} n \cdot T(n) &\leq 3n \cdot T(n-1) - 2n(n-1)T(n-2) \stackrel{!}{=} \frac{T(n)}{n!} \leq 3 \frac{T(n-1)}{(n-1)!} - 2 \frac{T(n-2)}{(n-2)!} \\ \frac{T(n)}{n!} \leq a_n &\Rightarrow a_n \leq 3a_{n-1} - 2a_{n-2} \dots \quad \text{مشتبه قابل} \rightarrow T(n) \leq n! (x_1^2 + x_2) \end{aligned}$$

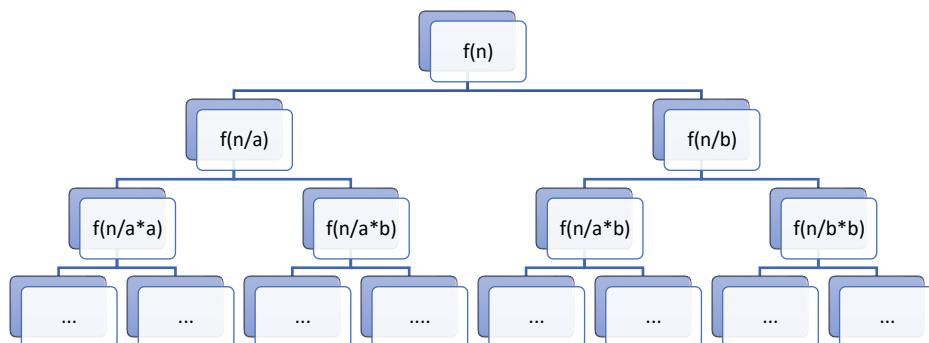
مثال : مطلوب است $a_0 = 1, a_1 = 2$. $T(n) = 2T(\frac{n}{2}) - 4T(\frac{n}{4}) + n(\log n)^2$ را بدست آورید.

$$\begin{aligned} 2^m = n &\Rightarrow m = \log n \Rightarrow T(2^m) \leq 2T(2^{m-1}) - 4T(2^{m-2}) + 2^m m^2 \rightarrow a_m \leq 2^m \\ &\Rightarrow a_m = 2a_{m-1} - 4a_{m-2} + 2^m m^2 \geq (x^2 - 4x + 4)(x-2)^3 \leq 0 \rightarrow 2, 2, 2, 2, 2 \\ a_m &\leq (x_1 + mx_2 + m^2 x_3 + m^3 x_4 + m^4 x_5) 2^m = \Theta(m^4 2^m) \\ &\Rightarrow T(n) \leq \Theta(\frac{\log n}{2^m} \cdot n) \end{aligned}$$

درخت بازگشت

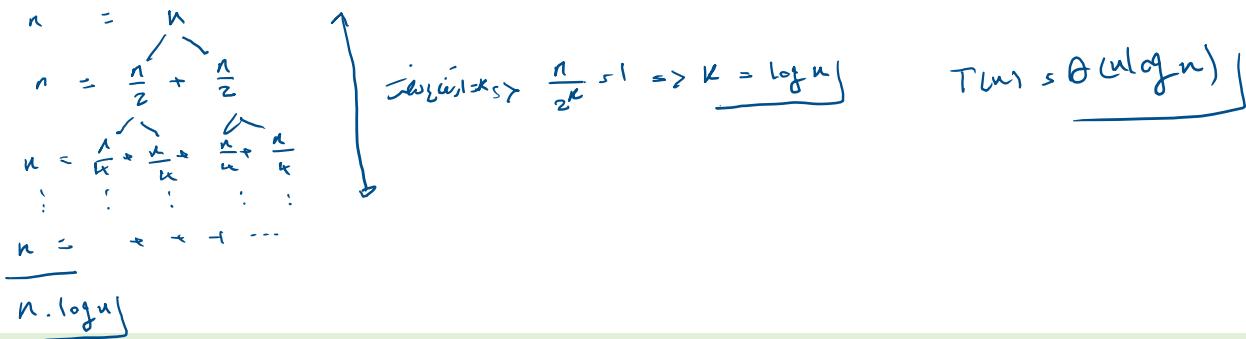
مثل همان روش جایگذاری از بالاست :

مثال : مطلوب است $T(n) = 2T(\frac{n}{a}) - 4T(\frac{n}{b}) + f(n)$ را بدست آورید.

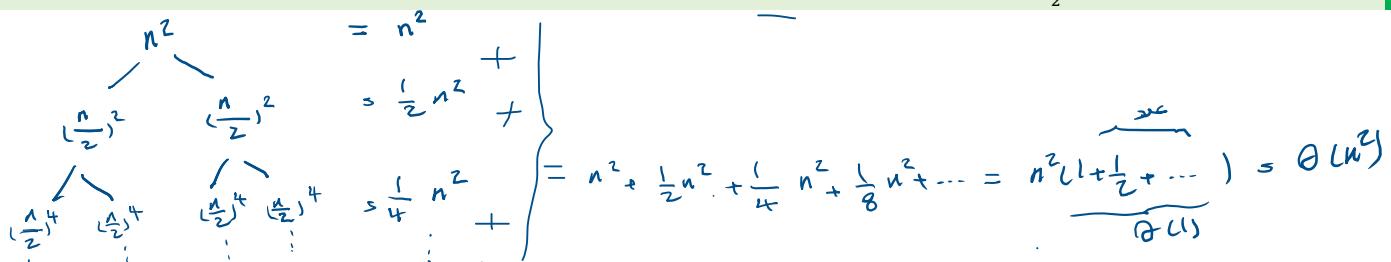


درخت را آنقدر گستردہ می کنیم تا به شرط اولیه برسیم.

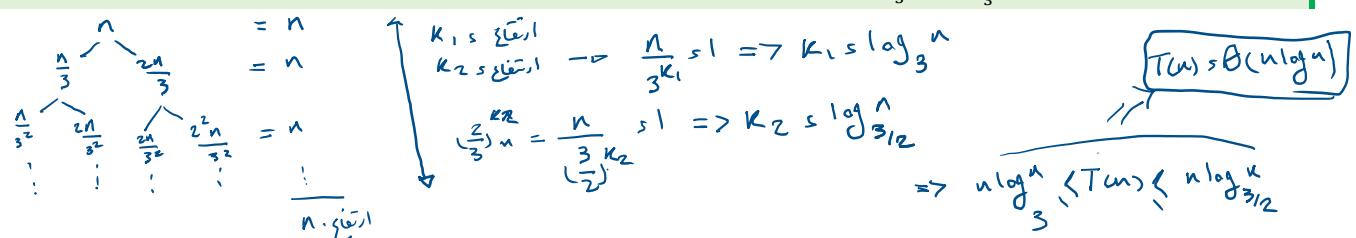
مثال : مطلوب است $T_{(n)} = 2T_{(\frac{n}{2})} + n$ را بدست آورید .



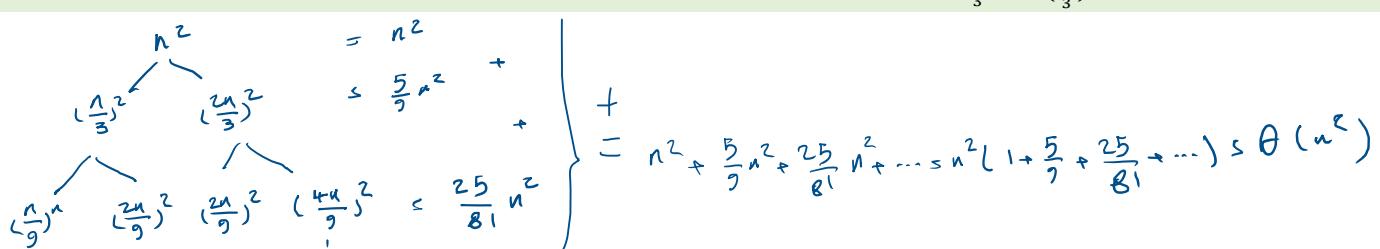
مثال : مطلوب است $T_{(n)} = 2T_{(\frac{n}{2})} + n^2$ را بدست آورید .



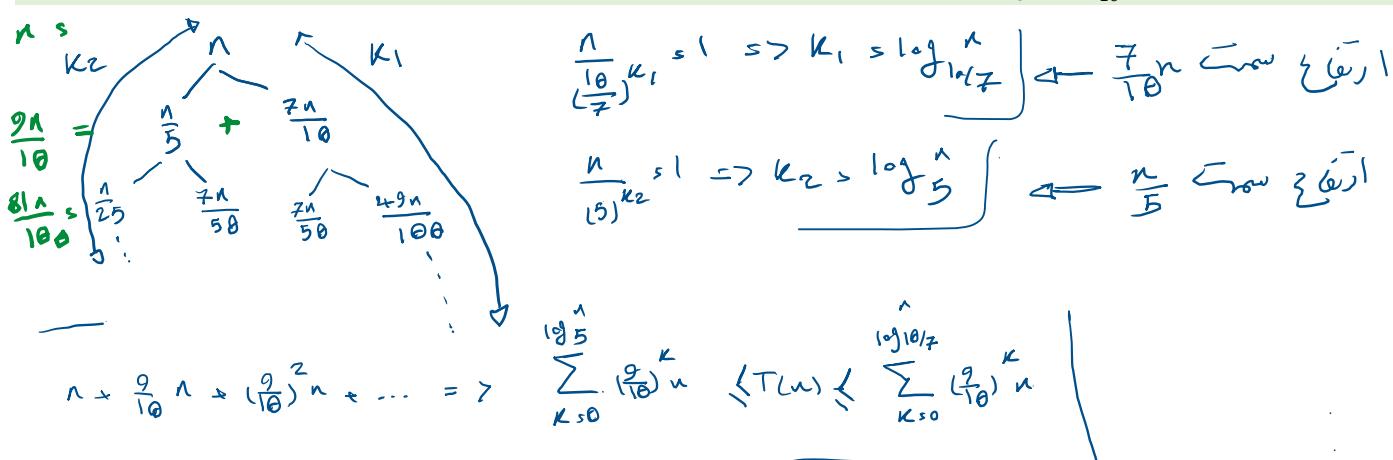
مثال : مطلوب است $T_{(n)} = T_{(\frac{n}{3})} + T_{(\frac{2n}{3})} + n$ را بدست آورید .



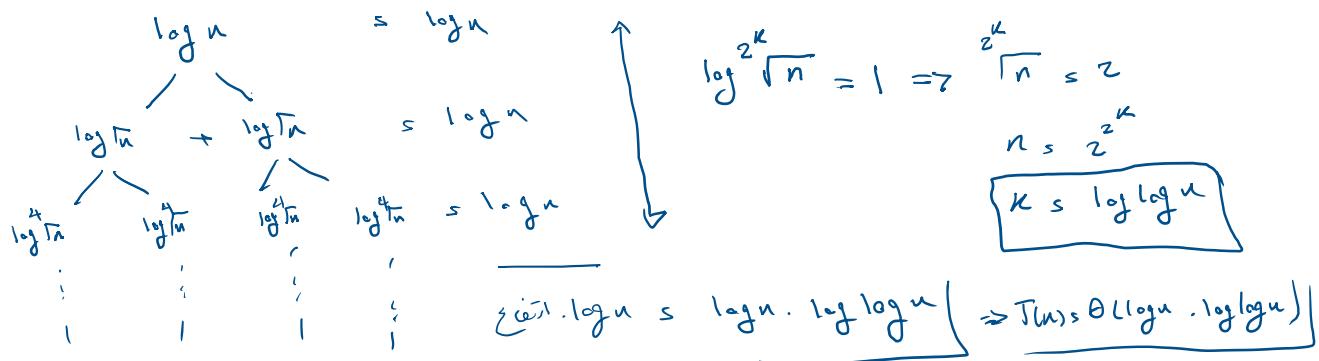
مثال : مطلوب است $T_{(n)} = T_{(\frac{n}{3})} + T_{(\frac{2n}{3})} + n^2$ را بدست آورید .



مثال : مطلوب است $T_{(n)} = T_{(\frac{n}{5})} + T_{(\frac{7n}{10})} + n$ را بدست آورید .



مثال : مطلوب است $T_{(n)} = 2T_{(\sqrt{n})} + \log n$ را بدست آورید .



قضیه master

$$T_{(n)} = aT\left(\frac{n}{b}\right) + f_{(n)}, \quad a, b = \text{constant} > 1$$

اگر $\epsilon > 0$ وجود داشته باشد :

1. if $f_{(n)} \in \Omega(n^{\log_b a + \epsilon}) \Rightarrow T_{(n)} \in \theta(f_{(n)})$
2. if $f_{(n)} \in O(n^{\log_b a - \epsilon}) \Rightarrow T_{(n)} \in \theta(n^{\log_b a})$
3. if $f_{(n)} \in \theta(n^{\log_b a}) \Rightarrow T_{(n)} \in \theta(f_{(n)} \cdot \log n)$

اگر $\epsilon > 0$ وجود نداشته باشد روش مستر جواب نمی دهد .

مثال : مطلوب است $n \log n = 2T_{(\frac{n}{2})} + n \cdot \log n$ را بدست آورید .

$$\log_2^2 1 = 1 \Rightarrow \begin{cases} 1 \in \Omega(m^{1+\epsilon}) \\ 1 \in O(m^{1-\epsilon}) \\ 1 \in \theta(m^1) \end{cases}$$

برای $n \log n$ نیز مطابق با این بُندهای ممکن است

با استفاده از master منطقی به جواب رسید .

تذکر : اگر $T_{(n)} \in \theta(n^{\log_b a} \cdot (\log n)^{k+1})$ باشد آنگاه $T_{(n)} = aT_{(\frac{n}{b})} + n^{\log_b a} \cdot (\log n)^k$ است مثل مثال بالا .

مثال : مطلوب است $T_{(n)} = 2T_{(\sqrt{n})} + \log n$ را بدست آورید . (مثالی که در روش درخت حل شد)

$$T_{(2^m)} \leq 2T_{(2^{\frac{m}{2}})} + m \rightarrow a_m \leq 2a_{\frac{m}{2}} + m$$

$* n \leq 2^m$

$$\log_2^2 1 = 1 \Rightarrow \begin{cases} 1 \in \Omega(m^{1+\epsilon}) \\ 1 \in O(m^{1-\epsilon}) \\ 1 \in \theta(m^1) \end{cases}$$

$\Rightarrow T(n) \in \Theta(\log n \cdot \log \log n)$

مثال : مطلوب است $T(n) = 2\sqrt{n}T(\sqrt{n}) + n \cdot \log n$ را بدست آورید . (مثالی که در روش درخت حل شد)

$$\frac{T(n)}{n} \leq \frac{2T(\sqrt{n})}{\sqrt{n}} + \log n \rightarrow a_n \leq 2a_{\sqrt{n}} + \log n \xrightarrow[n \leq 2^m]{n = 2^m} b_m \leq 2b_{\frac{m}{2}} + m$$

$b_m \in \Theta(m \log m)$

\downarrow

$T(n) \in \Theta(n \cdot \log n \cdot \log \log n) \leq$

$$\frac{T(n)}{n} \in \Theta(\log n \cdot \log \log n)$$

آرایه‌ها

مجموعه از خانه‌ها پشت سرهم حافظه که اسم آن آدرس اولین خانه می‌باشد و جنس تمامی خانه‌ها یکسان است.

► اگر داشته باشم $A[l_1 \dots l_2]$ آرایه‌ای دو بعد باشد و شروع آدرس از β باشد و اندازه هر داده n بایت باشد آنگاه خانه $[A[i]]$ آدرسش برابر است با :

- ذخیره سازی سطري : $address(A[i]) = ((i - l_1)).n + \beta$
- ذخیره سازی ستوني : $address(A[i]) = ((i - l_1)).n + \beta$

► اگر داشته باشم $A[u_1 \dots u_2][l_1 \dots l_2]$ آرایه‌ای دو بعد باشد و شروع آدرس از β باشد و اندازه هر داده n بایت باشد آنگاه خانه $[A[i][j]]$ آدرسش برابر است با :

- ذخیره سازی سطري : $address(A[i][j]) = ((i - l_1)(u_2 - u_1 + 1) + (j - u_1)).n + \beta$
- ذخیره سازی ستوني : $address(A[i][j]) = ((j - u_1)(l_2 - l_1 + 1) + (i - l_1)).n + \beta$

► اگر داشته باشم $A[o_1 \dots o_2][l_1 \dots l_2][u_1 \dots u_2]$ آرایه‌ای دو بعد باشد و شروع آدرس از β باشد و اندازه هر داده n بایت باشد آنگاه خانه $[A[i][j][k]]$ آدرسش برابر است با :

- ذخیره سازی سطري :

$$address(A[i][j][k]) = ((i - l_1)(u_2 - u_1 + 1)(o_2 - o_1 + 1) + (j - u_1)(o_2 - o_1 + 1) + (k - o_1)).n + \beta$$

- ذخیره سازی ستوني :

$$address(A[i][j][k]) = ((k - o_1)(u_2 - u_1 + 1)(l_2 - l_1 + 1) + (j - u_1)(l_2 - l_1 + 1) + (i - l_1)).n + \beta$$

► آرایه با سایر ابعاد نیز به صورت بالاست.

مثال : آرایه سه بعدی از اعداد صحیح int با ابعاد [10 ... 15][5 ... 50][3 ... 16] می‌باشد آدرس عنصر $A[10][15][5]$ را برای ترتیب سطري و ستونی بیابید اگر آدرس شروع ۵۰۰۰ باشد.

• ذخیره سازی سطري :

$$address(A[10][15][5]) = ((10 - 5)(16 - 3 + 1)(10 - 2 + 1) + (15 - 3)(10 - 2 + 1) + (5 - 2)).4 + 5000 = 3964$$

• ذخیره سازی سطري : جواب ۱۰۹۵۶

چند جمله ای

- به صورت $c_nx^n + c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + c_{n-3}x^{n-3} + \dots + c_1x + c_0$ می باشد.
- ضریب جملاتی که وجود ندارد صفر است.
- نمایش چند جمله ای روش اول (برای چند جمله ای هایی که ضریب صفر کمی دارند مناسب است)

ساختمان کلاس :

```
class Polynomial
{
private:
    double* Coef;
    int Max_Dgree = 0;
public:
    Polynomial(int max_dgree);
    ~Polynomial();
    bool AddPoly(double coef, int degree = 0);
    string To_string();
    string operator + (Polynomial& obj);
    double getCoef(int dgree);
    string operator * (Polynomial& obj);
    void zero();
    string RemoveZero(string str);
    double Cal(double num);
};
```

ساختمان کلاس : یک آرایه به اندازه بیشترین درجه چند جمله ای می سازیم

```
Polynomial::Polynomial(int max_dgree)
{
    Coef = new double[max_dgree + 1];
    Max_Dgree = max_dgree;
}
```

مخرب کلاس :

```
Polynomial::~Polynomial()
{
    delete[Max_Dgree] Coef;
}
```

برای افزودن یک term درجه آن و ضریب آن گرفته می شود که اگر درجه بیشتر از بیشترین درجه باشد خطای تولید می شود.

```
bool Polynomial::AddPoly(double coef, int degree)
{
    if(degree>Max_Dgree) throw exception((string("degree must be less than ")+to_string(Max_Dgree)).c_str());
    Coef[degree] = coef;
    return true;
}
```

تابع `getCoef` درجه می گیرد و ضریب آن را بازگشت می دهد.

```
double Polynomial::getCoef(int dgree)
{
    return this->Coef[dgree];
}
```

```
string Polynomial::RemoveZero(string str)
{
    while ((str.find('.') != -1 && str[str.length() - 1] == '0') || str[str.length() - 1] == '.')
    {
        str.erase(str.length() - 1, 1);
    }
    return str;
}
```

}

جمع دو چند جمله ای :

```

string Polynomial::operator + (Polynomial& obj) {

    if (this->Max_Dgree >= obj.Max_Dgree)
    {
        Polynomial* res = new Polynomial(this->Max_Dgree);
        for (int i = 0; i <= this->Max_Dgree; i++)
        {
            if (i <= obj.Max_Dgree)
            {
                res->AddPoly(this->Coef[i] + obj.Coef[i], i);
            }
            else {
                res->AddPoly(this->Coef[i], i);
            }
        }
        string ans = res->To_string();
        delete res;
        return ans;
    }
    else
    {
        Polynomial* res = new Polynomial(obj.Max_Dgree);
        for (int i = 0; i <= obj.Max_Dgree; i++)
        {
            if (i <= this->Max_Dgree)
            {
                res->AddPoly(this->Coef[i] + obj.Coef[i], i);
            }
            else {
                res->AddPoly(obj.Coef[i], i);
            }
        }
        string ans = res->To_string();
        return ans;
    }
}

```

ضرب دو چند جمله ای :

```

string Polynomial::operator * (Polynomial& obj)
{
    Polynomial pol = Polynomial(obj.Max_Dgree + this->Max_Dgree);
    pol.zero(); // این تابع تمامی عناصر را صفر میکند
    for (int i = 0; i <= obj.Max_Dgree; i++)
    {
        for (int j = 0; j <= this->Max_Dgree; j++)
        {
            pol.Coef[i + j] += obj.Coef[i] * this->Coef[j];
        }
    }
    return pol.To_string();
}

```

- نمایش چند جمله ای روش دوم (برای چند جمله ای هایی که ضریب صفر زیادی دارند مناسب است)
- یک ساختدار لازم است تا ضریب و قوه را نگه دارد.

برای چند جمله ای $1 + 2x^{1000}$ اگر از روش قبل استفاده کنیم می بایست ۱۰۰۰ خانه برای آرایه ایجاد کنیم که ۹۹۸ خانه بلا استفاده می ماند برای جلوگیری از این موضوع یک ساختدار تعریف میکنیم و تعداد جملات غیر صفر را از کاربر میگیریم.

کد قبل را به شکل زیر تغییر میدهیم : توابع تغییرات اندکی میکند برای جلوگیری از افزایش صفحات مجدد آورده نشده است.

```
struct term {
    float coef;
    int exp;
};

class Polynomial
{
private:
    term* Coef;
    int MaxNonZeroTerm = 0;
public:
    Polynomial(int MaxNonZeroTerm);
    ~Polynomial();
    bool AddPoly(double coef, int degree = 0);
    string To_string();
    string operator + (Polynomial& obj);
    double getCoef(int dgree);
    string operator * (Polynomial& obj);
    void zero();
    string RemoveZero(string str);
    double Cal(double num);
};

};
```

ماتریس و ماتریس پراکنده (Sparse Matrix)

- ماتریس خلوت (Sparse matrix) ماتریسی است که اکثر درایه های آن صفر هستند.
- ماتریس خلوت (Sparse matrix) ماتریسی است که اکثر درایه های آن صفر هستند.

	row	column	value
0	0	1	1
0	0	2	5
1	1	0	3
2	2	2	7
2	2	4	9
3	3	3	4
4	4	1	2
4	4	4	8

- برای ذخیره سازی ماتریس شماره ردیف ، شماره ستون و مقدار خانه های غیر صفر نگه داری می شود در هر لحظه از ذخیره سازی ماتریس خلوت باید اول بر اساس ردیف داده ها صعوادی باشند و بعد بر اساس ردیف صعوادی باشند.
- برای ذخیره سازی هر عنصر ماتریس خلوت که شما ردیف و ستون و مقدار است از ساختمان زیر استفاده می کنیم:

```

1. struct SparstMatixTerm
2. {
3.     int row;
4.     int clm;
5.     int value;
6. };

```

- در کلاس ماتریس خلوت داریم :

```

1. class SparstMatix
2. {
3. private:
4.     SparstMatixTerm* SparstMatixTerms;
5.     int SparstMatixTermsCount;
6.     int index = 0;
7. public:
8.     SparstMatix(int SparstMatixTermsCount);
9.     ~SparstMatix();
10.    void Insert(int row , int clm,int value);
11.    SparstMatix* operator + (SparstMatix& sm);
12.    SparstMatix* operator * (SparstMatix& sm);
13.    SparstMatix* Transpose();
14. };

```

- سازنده و مخرب کلاس :

```

1. SparstMatix::SparstMatix(int SparstMatixTermsCount) :SparstMatixTermsCount(SparstMatixTermsCount)
2. {
3.     SparstMatixTerms = new SparstMatixTerm[SparstMatixTermsCount];
4. }
5. SparstMatix::~SparstMatix()
6. {
7.     delete [SparstMatixTermsCount]SparstMatixTerms;
8. }

```

افزودن یک term جدید : ➤

```
1. void SparstMatix::Insert(int row , int clm, int value)
2. {
3.     if(index>SparstMatixTermsCount) throw exception();
4.     SparstMatixTerm sp{row,clm,value};
5.     SparstMatixTerms[index++] = sp;
6. }
```

ترانهاده : ➤

- روش های متفاوتی برای این کار وجود دارد مثلا بباید ردیف ها را از بالا پیمایش کنیم و ترانهاده کنیم و
- در روشی که ذکر می شود در داخل روش از مرتب سازی counting استفاده شده است .

row	col	value	row	col	value	index
b[0]			a[0]	0	0	15
[1]			[1]	0	4	91
[2]			[2]	1	1	11
[3]			[3]	2	1	3
[4]			[4]	2	5	28
[5]			[5]	3	0	22
[6]			[6]	3	2	-6
[7]			[7]	5	0	-15
index	[0]	[1]	[2]	[3]	[4]	[5]
RowSize	3	2	1	0	1	1
RowStart	0	3	5	6	6	7

در شکل مقابل rowsize یعنی ستون چه تعداد عنصر داریم که تعداد آنها index آرایه است مثلا ما ۳ عنصر با ستون ۰ داریم یا ۲ عنصر با ستون ۱ و ... rowstart یعنی index نوشتمن عنصر در ماتریس خلوت چیست که هر عنصر آن مجموع قبلى است مثلا برای $index = 2$ مقدار rowstart برابر است با $2 + 3 = 5$ و ... با استفاده از rowstart میفهمیم که ستون های مثلا صفر باید از خانه صفر ام تا ۳ ام آرایه قرار بگیرند به پیاده سازی دقت کنید .

پیچیدگی زمانی ترانهاده برابر است با : $T(n) \in \theta(terms)$

```
SparstMatix* SparstMatix::Transpose()
{
    int *rowSize = new int[SparstMatixTermsCount];
    int *rowStart = new int[SparstMatixTermsCount];
    for (size_t i = 0; i < SparstMatixTermsCount; i++)
        rowSize[i] = 0;
    for (size_t i = 0; i < SparstMatixTermsCount; i++)
        rowSize[SparstMatixTerms[i].clm]++;
    rowStart[0] = 0;
    for (size_t i = 1; i < SparstMatixTermsCount; i++)
        rowStart[i] = rowStart[i-1]+rowSize[i-1];

    SparstMatix *res = new SparstMatix(SparstMatixTermsCount);
    for (size_t i = 0; i < SparstMatixTermsCount; i++)
    {
        int j = rowStart[SparstMatixTerms[i].clm];
        res->SparstMatixTerms[j].clm = SparstMatixTerms[i].row;
        res->SparstMatixTerms[j].row = SparstMatixTerms[i].clm;
        res->SparstMatixTerms[j].value = SparstMatixTerms[i].value;
        rowStart[SparstMatixTerms[i].clm]++;
    }
    return res;
}
```

➢ جمع : برای جمع باید درایه های نظیر به نظیر را باهم جمع کنیم البته ممکن است در اثر جمع برخی درایه ها صفر شوند که باید حذف شوند . اگر ستون ها و ردیف ها برابر بود باهم جمع می شوند در غیر این صورت ماتریسی که ردیف کوچکتری دارد نوشته می شود ولی اگر ردیف ها برابر باشد ماتریسی که ستون کوچکتری داشته باشد در خروجی نوشته می شود .

$$T(n) \in \theta(A.terms + B.terms)$$

```

1.   SparstMatix* SparstMatix::operator + (SparstMatix& sm)
2.   {
3.     int pos1 = 0 , pos2 = 0;
4.     SparstMatix *res = new SparstMatix(Max(SparstMatixTermsCount,sm.SparstMatixTermsCount));
5.     while (pos1 < this->SparstMatixTermsCount && pos2 < sm.SparstMatixTermsCount)
6.     {
7.       if((this->SparstMatixTerms[pos1].row > sm.SparstMatixTerms[pos2].row)
8.          || (this->SparstMatixTerms[pos1].row == sm.SparstMatixTerms[pos2].row
9.             && this->SparstMatixTerms[pos1].row > sm.SparstMatixTerms[pos2].row))
10.      {
11.        res->Insert(sm.SparstMatixTerms[pos2].row,
12.                      sm.SparstMatixTerms[pos2].clm,
13.                      sm.SparstMatixTerms[pos2].value);
14.        pos2++;
15.      } else if((this->SparstMatixTerms[pos1].row < sm.SparstMatixTerms[pos2].row)
16.          || (this->SparstMatixTerms[pos1].row == sm.SparstMatixTerms[pos2].row
17.             && this->SparstMatixTerms[pos1].row < sm.SparstMatixTerms[pos2].row))
18.      {
19.        res->Insert(this->SparstMatixTerms[pos1].row,
20.                      this->SparstMatixTerms[pos1].clm,
21.                      this->SparstMatixTerms[pos1].value);
22.        pos1++;
23.      }
24.    else
25.    {
26.      int sum = this->SparstMatixTerms[pos1].value+sm.SparstMatixTerms[pos2].value;
27.      if(!sum)
28.      {
29.        res->Insert(this->SparstMatixTerms[pos1].row,
30.                      this->SparstMatixTerms[pos1].clm,sum);
31.        pos1++;
32.        pos2++;
33.      }
34.    }
35.  }
36.  while (pos1<SparstMatixTermsCount)
37.  {
38.    res->Insert(this->SparstMatixTerms[pos1].row,this->SparstMatixTerms[pos1].clm,this->SparstMatixTerms[pos1].value);
39.    pos1++;
40.  }
41.  while (pos2<SparstMatixTermsCount)
42.  {
43.    res->Insert(sm.SparstMatixTerms[pos2].row,sm.SparstMatixTerms[pos2].clm,sm.SparstMatixTerms[pos2].value);
44.    pos2++;
45.  }
46.  return res;
47. }
```

✖ ضرب :

- برای ضرب دو ماتریس A و B
 - ترانهاده ماتریس B را حساب میکنیم و 'B' مینامیم.
 - به صورت سطری در ماتریس A و 'B' پیمایش میکنیم.
 - برای هر سطر ماتریس A ؛ تمام سطرهای B' یک به یک طی میکنیم.
 - درایه ها با شماره ستون برابر را با هم ضرب کرده و پس از جمع در ماتریس پاسخ ثبت میکنیم.
- به طور خلاصه ضرب دو ماتریس خلوت به این گونه است که تمام درایه ها با شماره ستون برابر در A و B' دو به دو در یکدیگر ضرب میشوند و درایه هایی که شماره ردیفشان برابر است را با هم جمع میکنیم.

$$T(n) \in \theta(B.\text{column} \times A.\text{terms} + A.\text{row} \times B.\text{terms})$$

```

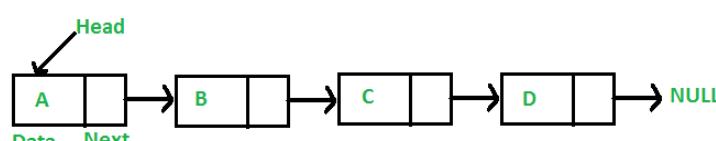
1.   SparstMatix* SparstMatix::operator * (SparstMatix& sm)
2. {
3.     SparstMatix* _B = sm.Transpose();
4.     int pos1 = 0 , pos2 = 0;
5.     SparstMatix *res = new SparstMatix(Max(SparstMatixTermsCount,sm.SparstMatixTermsCount));
6.     for (pos1 = 0; pos1 < SparstMatixTermsCount; )
7.     {
8.       int r = this->SparstMatixTerms[pos1].row;
9.       for (pos2 = 0; pos2 < sm.SparstMatixTermsCount; )
10.      {
11.        int c = sm.SparstMatixTerms[pos2].clm;
12.        int tmp1 = pos1,tmp2 = pos2,sum = 0;
13.        while (tmp1 < SparstMatixTermsCount && this->SparstMatixTerms[tmp1].row == r
14.               && tmp2 < sm.SparstMatixTermsCount && sm.SparstMatixTerms[tmp2].clm == c )
15.        {
16.          if(this->SparstMatixTerms[tmp1].clm < sm.SparstMatixTerms[tmp2].clm)
17.            tmp1++;
18.          else if (this->SparstMatixTerms[tmp1].clm > sm.SparstMatixTerms[tmp2].clm)
19.            tmp2++;
20.          else
21.            sum += this->SparstMatixTerms[tmp1].value * this->SparstMatixTerms[tmp1].value;
22.        }
23.        if(!sum)
24.        {
25.          res->Insert(r,c,sum);
26.        }
27.        while (pos2 < sm.SparstMatixTermsCount && sm.SparstMatixTerms[pos2].clm == c)
28.          pos2++;
29.      }
30.      while (pos1 < SparstMatixTermsCount && SparstMatixTerms[pos1].row == 3)
31.        pos1++;
32.    }
33.    return res;
34. }
```

لیست پیوندی

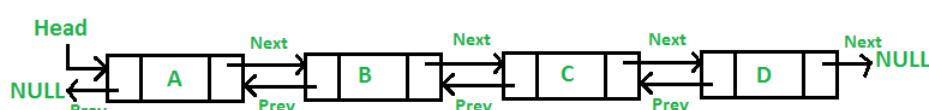
زمانی برای استفاده مناسب است که تعداد عناصر را ندانیم چرا که طول لیست بخلاف آرایه تغییر میکند، هر گره (Node) در لیست پیوندی حداقل دارای یک خانه برای مقدار و یک خانه برای ذخیره سازی آدرس عنصر بعدی یا قبلی است. به تعداد عناصر موجود حافظه می‌گیرد، اما بخلاف آرایه امکان دسترسی مستقیم به هر عنصر نیست.

Head: همیشه به عنصر اول لیست اشاره میکند.

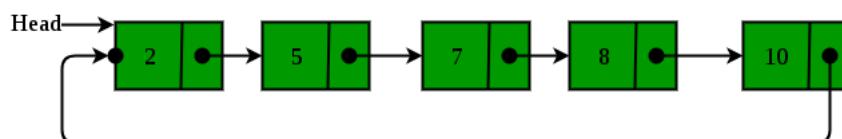
۱- لینک لیست یک طرفه:



۲- لینک لیست دو طرفه:



۳- لینک لیست حلقوی:



مثال: یک لیست پیوندی ۱۰۰ گره تعریف کنید که اعداد ۱ تا ۱۰۰ را در آن ذخیره کند.

```
struct Node // تعریف گره
{
    int Value;
    Node* next;
};

int main()
{
    Node* head;
    head = new Node();
    head->Value = 1;
    Node* temp = head;
    for (int i = 2; i <= 100; i++)
    {
        Node* newNode = new Node;
        newNode->Value = i;
        temp->next = newNode;
        temp = temp->next ;
    }
    temp->next = nullptr;
}
```

مثال : دستوری بنویسید یک Node بعد از x درج کند.

```
Node* y = new Node();
y->next = x->next;
x->next = y;
```

مثال : دستوری بنویسید یک Node با آدرس x حذف کند.

```
Node* y = head;
while (y->next != x)
    y = y->next;
y->next = y->next->next; //or y->next = x->next;
delete x;
```

مثال : دستوری بنویسید یک Node بعد از x در لیست دو طرفه درج کند.

```
struct Node // تعریف گره
{
    int Value;
    Node* next;
    Node* previous ;
};

Node* y = new Node();
y->next = x->next;
x->next->previous = y;
x->next = y;
y->previous = x;
```

مثال : دستوری بنویسید یک Node با آدرس x در لیست دو طرفه حذف کند.

```
x->previous->next = x->next;
x->next->previous = x->previous;
```

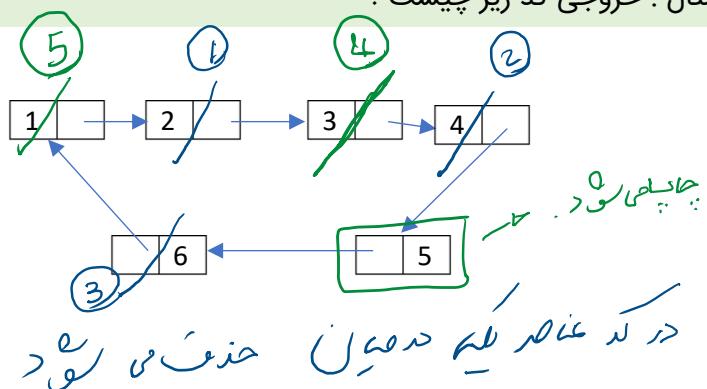
مثال : دستوری بنویسید یک Node به اول لیست حلقوی درج کند.

```
Node* End = head;
Node* x = new Node();
while (End->next = head)
    End->next = End;
End->next = x;
x->next=head;
head=x;
```

بهتر از برای لینک لیست حلقوی بجای آدرس اولین خانه آخرین خانه نگهداری شود.

```
Node* x = head
while (x->next != x)
{
    x->next = x->next->next;
    x=x->next;
}
cout << x->Value;
```

مثال : خروجی کد زیر چیست ؟

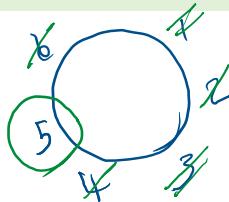


بیاده سازی لینک لیست را میتوانید در گیت هاب مشاهده کنید.

مسئله جوزف (Josephus)

اعداد اتا n را دور یک میز می‌چینیم و درجهٔ عقربه‌های ساعت حرکت می‌کنیم، از شمارهٔ ۲ اعداد را یک درمیان حذف می‌کنیم، آخرين عدد را $f(n)$ می‌نامیم این عدد کدام است؟

مثال : مثال بالا را بررسی می‌کنیم :



رابطه بازگشتی برای مسئله جوزف برقرار است :

$$\begin{cases} f_{(2n)} = 2f_{(n)} - 1 \\ f_{(2n+1)} = 2f_{(n)} + 1 \\ f_{(1)} = 1, f_{(2)} = 1 \end{cases}$$

مثلا برای $n=6$ داریم

$$f_{(6)} = 2f_{(3)} - 1 = 2 * 6 - 1 = 5$$

$$f_{(3)} = 2f_{(1)} + 1 = 3$$

مثال : مسئله جوزف را برای $n=75$ حساب بررسی کنید.

$$f_{(75)} = 2f_{(37)} + 1 = 2 * 11 + 1 = 23$$

$$f_{(37)} = 2f_{(18)} + 1 = 2 * 5 + 1 = 11$$

$$f_{(18)} = 2f_{(9)} - 1 = 2 * 3 - 1 = 5$$

$$f_{(9)} = 2f_{(4)} + 1 = 2 * 1 + 1 = 3$$

$$f_{(4)} = 2f_{(2)} - 1 = 1$$

رابطه بازگشتی بالا برای n ها بزرگ بسیار سخت محاسبه می‌شود از یک تکنیک استفاده می‌کنیم :

عدد را به باینری مینویم یک شیفت چرخشی درجهٔ عقربه‌های ساعت می‌دهیم .

$$(75)_{10} = (1001011)_2 \xrightarrow{\text{cslsh}} (00010111)_2 = (23)_{10}, \quad (6)_{10} = (110)_2 \xrightarrow{\text{cslsh}} (101)_2 = (5)_{10}$$

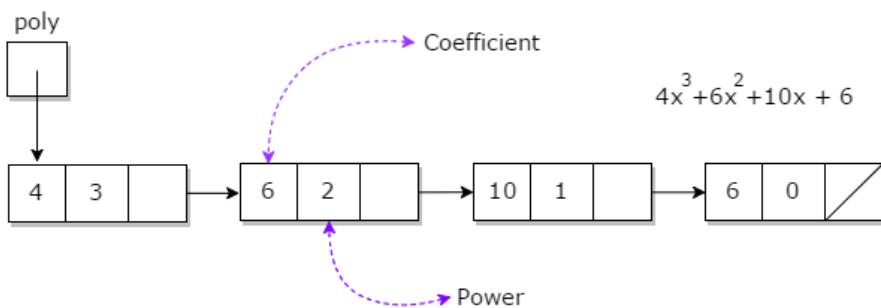
تبديل اعداد به باینری ممکن است دشوار باشد از این رو اگر رابطه بازگشتی مسئله جوزف را حل کنیم به عبارت زیر می‌رسیم در واقع رابطه بالا که به صورت باینری گفته شد یعنی عدد را از توان دو قبل کم کنیم ، ضرب در ۲ کنیم و با یک جمع کنیم

$$f_{(n)} = 2(n - 2^{\lfloor \log_2 n \rfloor}) + 1$$

مثال : مسئله جوزف را برای $n=1000$ حساب بررسی کنید.

$$f_{(1000)} = 2(1000 - 512) + 1 = 977$$

لیست پیوندی - چند جمله ای



ساختار هر Node :

```
struct Node
{
    double coeff;
    int pow_val;
    Node* next;
};

class polynomial
{
private:
    Node* head = nullptr;
public:
    void Insert(double coeff , int power);
    string ToString();
    polynomial* operator +(polynomial* poly);
    polynomial* operator *(polynomial* poly);
    void simplify(polynomial* node);
    void Delete(Node* poly);
};
```

ساختار کلاس چند جمله ای :

```
void polynomial::Insert(double coeff , int power)
{
    Node* node = new Node();
    node->coeff = coeff;
    node->pow_val=power;
    node->next = nullptr;
    if(head == nullptr)
        head = node;
    else
    {
        Node* tmp = head;
        while (tmp->next != nullptr)
            tmp = tmp->next;
        tmp->next = node;
    }
}
```

افزودن یک Term :

تبدیل چند جمله ای به : string

```
string polynomial::ToString()
{
    string result = "Empty";
    if(head!=nullptr)
    {
        result = "";
        Node* tmp = head;
        while (tmp->next !=nullptr)
        {
            result += to_string(tmp->coeff) + "x^" + to_string(tmp->pow_val);
            tmp = tmp->next;
            if(tmp->next != nullptr)
                result += " + ";
        }
    }
    return result;
}
```

جمع دو چند جمله ای :

```
polynomial* polynomial::operator +(polynomial* poly)
{
    polynomial* result;
    Node* start1 = this->head;
    Node* start2 = poly->head;
    while (start1->next && start2->next ) // or start1->next != nullptr && start2->next != nullptr
    {
        if(start1->pow_val > start2->pow_val)
        {
            result->Insert(start1->coeff,start1->pow_val);
            start1 = start1->next;
        }
        else if(start1->pow_val < start2->pow_val)
        {
            result->Insert(start2->coeff,start2->pow_val);
            start2 = start2->next;
        }
        else
        {
            if(start2->coeff + start1->coeff != 0)
                result->Insert(start2->coeff + start1->coeff,start2->pow_val);
            start2 = start2->next;
            start1 = start1->next;
        }
    }
    while (start1->next || start2->next) {
        if (start1->next) {
            result->Insert(start1->coeff,start1->pow_val);
            start1 = start1->next;
        }
        if (start2->next) {
            result->Insert(start2->coeff,start2->pow_val);
            start2 = start2->next;
        }
    }
}
```

ضرب دو چند جمله ای :

```
polynomial* polynomial::operator *(polynomial* poly)
{
    polynomial* result;
    Node* start1 = this->head;
    Node* start2 = poly->head;
    while (start1->next)
    {
        start2 = poly->head;
        while (start2->next)
        {
            int coeff = start1->coeff * start2->coeff;
            int power = start1->pow_val + start2->pow_val;
            result->Insert(coeff, power);
            start2 = start2->next;
        }
        start1 = start1->next;
    }
    simplify(result);
    return result;
}
```

ساده سازی حاصل ضرب :

```
void polynomial::simplify(polynomial* poly)
{
    Node *ptr1, *ptr2;
    ptr1 = poly->head;
    /* Pick elements one by one */
    while (ptr1 != nullptr && ptr1->next != nullptr) {
        ptr2 = ptr1;
        // Compare the picked element with rest of the elements
        while (ptr2->next != nullptr) {
            ptr2 = ptr2->next;
            // If power of two elements are same
            if (ptr1->pow_val == ptr2->pow_val) {

                // Add their coefficients and put it in 1st element
                ptr1->coeff = ptr1->coeff + ptr2->coeff;
                // remove the 2nd element
                Delete(ptr2);
            }
            else
                ptr2 = ptr2->next;
        }
        ptr1 = ptr1->next;
    }
}
```

: Node یک حذف

```
void polynomial::Delete(Node* node)
{
    Node* tmp = this->head;
    while (tmp->next != node)
        tmp = tmp->next;
    tmp->next = node->next;
    delete node;
}
```

جست و جو در آرایه (Liner Search)

پیچیدگی زمانی

۱- بهترین حالت : $T_{(n)best} = \theta(1)$

۲- حالت میانگین : $T_{(n)average} = \theta(n)$

۳- بدترین حالت : $T_{(n)average} = \theta(n)$

```

int search(int arr[], int n, int x)
{
    int i;
    for (i = 0; i < n; i++)
        if (arr[i] == x)
            return i;
    return -1;
}

```

مثال : فرض کنید x به احتمال ۵۰ درصد در آرایه وجود دارد ، به طور متوسط چه تعداد عنصر برای یافتن x بررسی می شود؟

$$\text{ave}_s = \sum \frac{P_i}{n} \times \text{امضات} \Rightarrow \text{ave}_s = \frac{\frac{P}{n} + \frac{P}{n} \times 2 + \dots + \frac{P}{n} \times (n-1)}{n}$$

حکم $\frac{P}{n}$ در حالت ایم برای $\frac{P}{n}$ است
عد حکم $\frac{P}{n}$ در حالت $\frac{P}{n}$ است
 $\frac{P}{n}$ کلیه نتایج

$\Rightarrow \text{ave}_s = \frac{3n}{4}$

جست و جوی دودویی (برای آرایه های مرتب شده)

پیچیدگی زمانی

۱- بهترین حالت : $T_{(n)best} = \theta(1)$

۲- حالت میانگین : $T_{(n)average} = O(\log(n))$

۳- بدترین حالت : $T_{(n)average} = O(\log(n))$

بازگشتی :

```

int binarySearch(int arr[], int l, int r, int x)
{
    if (r >= l) {
        int mid = l + (r - l) / 2;

        if (arr[mid] == x)
            return mid;

        if (arr[mid] > x)
            return binarySearch(arr, l, mid - 1, x);
        else
            return binarySearch(arr, mid + 1, r, x);
    }
    return -1;
}

```

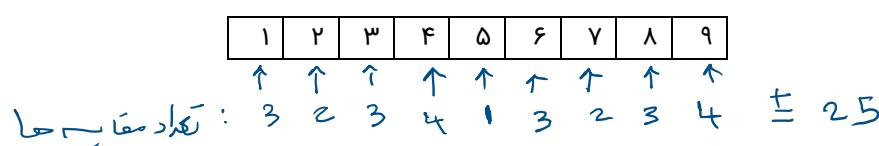
غیر بازگشتی :

```

int binarySearch(int arr[], int l, int r, int x)
{
    while (l <= r)
    {
        int m = l + (r-1)/2;
        if (arr[m] == x)
            return m;
        if (arr[m] < x)
            l = m + 1;
        else
            r = m - 1;
    }
    return -1;
}

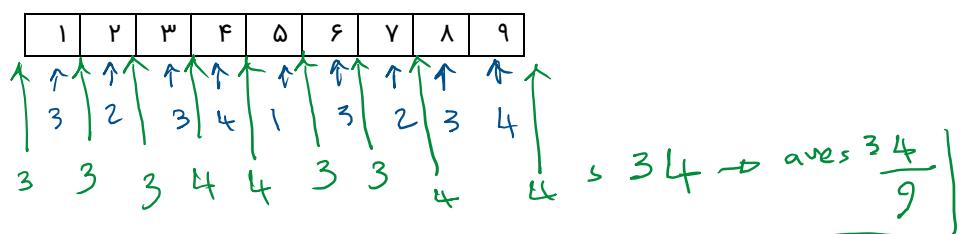
```

مثال : یک آرایه ۹ تایی داریم مطلوب است تعداد مقایسه های موفق جست و جوی دودویی ؟



۲۵ $\leq \frac{25}{9}$ \rightarrow تعداد متوسط مقایسه ها

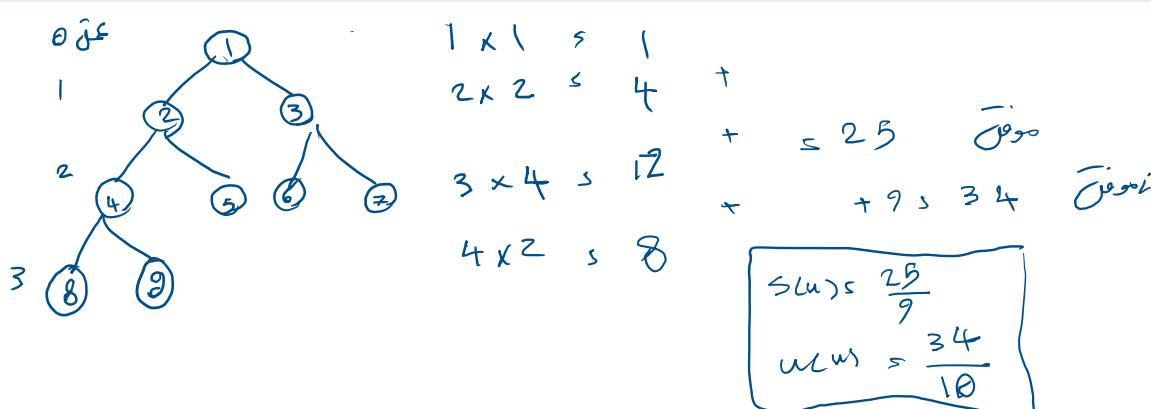
مثال : یک آرایه ۹ تایی داریم مطلوب است تعداد مقایسه های ناموفق جست و جوی دودویی ؟



اگر عناصر یک آرایه را مرتب شده را به صورت درخت کامل دودویی رسم کنیم

۱. تعداد جست و جو های موفق = $(\text{عمق}+1) * \text{تعداد گره های عمق}$
۲. تعداد جست و جو های ناموفق = موفق + تعداد عناصر آرایه
۳. تعداد جست و جو های متوسط موفق (S) = $\text{تعداد جست و جو های موفق} / \text{تعداد عناصر آرایه}$
۴. تعداد جست و جو های متوسط ناموفق (U) = $\text{تعداد جست و جو های ناموفق} / \text{تعداد عناصر آرایه} + 1$
۵. رابطه بین S, U به صورت $S = nS(n) + n(n+1)U(n)$ می باشد.

مثال : یک آرایه ۹ تایی داریم مطلوب است تعداد مقایسه های متوسط موفق و ناموفق جست و جوی دودویی ؟



$$T_{(n)} = T_{(\lfloor \frac{n}{2} \rfloor)} + 1 = \lfloor \log n \rfloor + 1 = \lfloor \log(n+1) \rfloor$$

حداکثر تعداد مقایسه های جست و جو دودویی

- در جست جو دودویی حداقل مقایسه موفق ۱ و حداکثر $\lfloor \log n \rfloor + 1$
- در جست جو دودویی حداقل مقایسه ناموفق $\lfloor \log n \rfloor$ و حداکثر $\lfloor \log n \rfloor + 1$

مثال : فرض کنید آرایه ای داریم به طول n ولی ما n را نمی شناسیم . یعنی اندازه آرایه را نمیدانیم . این آرایه به صورت صعودی مرتب است ، جست جو x در آرایه بهتر است از چه روشی باشد و از چه مرتبه ای است.

مثال : آرایه ای داریم به طول مشخص n ، این آرایه تا یک نقطه نامشخص صعودی است و از آن به بعد نزولی است جست و جو x در آرایه به چه صورت است ؟

مثال : جست و جو x را در آرایه چرخشی بررسی کنید.

مثال : در آرایه n تایی جست و جو ماکسیمم محلی چگونه است؟

مثال : ۳۱ جعبه کنار هم هست که هر جعبه شامل عددی است که وقتی در جعبه باز می شود ، آن عدد معلوم است . می خواهیم یک جعبه پیدا کنیم که عدش از مجاورش بیشتر یا مساوی باشد . در بدترین حالت در چند جعبه را بازکنیم.

جست و جو درون یاب (Interpolation search)

- بهبود یافته جست و جو دودویی است و برای آرایه ها با توزیع غیر یکنواخت مناسب تر است.

➤ پیچیدگی زمانی

➤ ۱- بهترین حالت : $T_{(n)best} = \theta(1)$

➤ ۲- حالت میانگین: $T_{(n)average} = \theta(\log(\log(n)))$

۳- بدترین حالت: $T_{(n)average} = \theta(n)$



غیر بازگشتی:

```
int interpolationSearch(int arr[], int n, int x)
{
    // Find indexes of two corners
    int lo = 0, hi = (n - 1);
    // Since array is sorted, an element present
    // in array must be in range defined by corner
    while (lo <= hi && x >= arr[lo] && x <= arr[hi])
    {
        if (lo == hi)
        {
            if (arr[lo] == x) return lo;
            return -1;
        }
        // Probing the position with keeping
        // uniform distribution in mind.
        int pos = lo + (((double)(hi - lo) /
        (arr[hi] - arr[lo])) * (x - arr[lo]));

        // Condition of target found
        if (arr[pos] == x)
            return pos;

        // If x is larger, x is in upper part
        if (arr[pos] < x)
            lo = pos + 1;
        // If x is smaller, x is in the lower part
        else
            hi = pos - 1;
    }
    return -1;
}
```

بازگشتی:

```
int interpolationSearch(int arr[], int lo, int hi, int x)
{
    int pos;

    // Since array is sorted, an element present
    // in array must be in range defined by corner
    if (lo <= hi && x >= arr[lo] && x <= arr[hi]) {

        // Probing the position with keeping
        // uniform distribution in mind.
        pos = lo + (((double)(hi - lo) / (arr[hi] - arr[lo])) * (x - arr[lo]));

        // Condition of target found
        if (arr[pos] == x)
            return pos;

        // If x is larger, x is in right sub array
        if (arr[pos] < x)
            return interpolationSearch(arr, pos + 1, hi, x);

        // If x is smaller, x is in left sub array
        if (arr[pos] > x)
            return interpolationSearch(arr, lo, pos - 1, x);
    }
    return -1;
}
```

جست و جو ترکیبی (Hybrid search)

اگر تعداد عناصر آرایه کمتر از ۱۶ باشد جست و جوی خطی نتیجه خوبی می دهد

```
int hybrid_search( Type obj, Type array[], int first, int last) {
    int use_binary_search = false;
    while (last - first > 16 ) {
        int midpoint = (first + last)/2; // point of binary search
        int b = use_binary_search ? midpoint : ((last - first)*(obj- array[first])) / (array[last] - array[first]);
        if ( obj == array[b] ) {
            return b;
        } else if ( obj < array[b] ) {
            last = b - 1;
            use_binary_search = ( midpoint < b );
        } else {
            first = b + 1;
            use_binary_search = ( midpoint > b );
        }
    }
    return linear_search( obj, array, first, last );
}
```

جست و جو ترکیبی (conditional search)

```
int binary_search( int obj, int array[], int low, int high) {
    while ( high - low > 16 ) {
        int mid = (low + high) /2;
        if ( obj == array[mid] ) {
            return mid;
        } else if ( obj < array[mid] ) {
            high = mid - 1;
        } else {
            low = mid + 1;
        }
    }
    return linear_search( obj, array, low, high );
}
```

جست و جو در لیست پیوندی

۱- جست و جو خطی

```
bool search(Node* head, int x)
{
    Node* current = head;
    while (current != NULL)
    {
        if (current->Value == x)
            return true;
        current = current->next;
    }
    return false;
}
```

۲- جست و جو دودویی

```
struct Node* middle(Node* start, Node* last)
{
    if (start == NULL)
        return NULL;

    struct Node* slow = start;
    struct Node* fast = start -> next;

    while (fast != last)
    {
        fast = fast -> next;
        if (fast != last)
        {
            slow = slow -> next;
            fast = fast -> next;
        }
    }

    return slow;
}
Node* binarySearch(Node *head, int value)
{
    struct Node* start = head;
    struct Node* last = NULL;

    do
    {
        // Find middle
        Node* mid = middle(start, last);

        if (mid == NULL)
            return NULL;

        // If value is present at middle
        if (mid -> Value == value)
            return mid;

        // If value is more than mid
        else if (mid -> Value < value)
            start = mid -> next;

        // If the value is less than mid.
        else
            last = mid;
    } while (last == NULL ||
             last != start);

    // value not present
    return NULL;
}
```

مرتب سازی

- (۱) مرتب سازی داخلی : مرتب سازی عناصر حافظه اصلی
- (۲) مرتب سازی خارجی : مرتب سازی عناصر حافظه کمکی (Disk)
- (۳) مرتب سازی درجا(in place) : حافظه کمکی ثابت (مثلاً یک یا دو متغیر کمکی نیاز نداشت)
- (۴) مرتب سازی غیر درجا(out of place) : نیاز به حافظه کمکی به اندازه حافظه وابسته است
- (۵) مرتب سازی پایدار(stable) : ترتیب عناصر یکسان ثابت است
- (۶) مرتب سازی ناپایدار(unstable) : ترتیب عناصر یکسان لزوماً ثابت نیست
۱. تمامی الگوریتم های مرتب سازی را میتوان بدون افزایش مرتبه زمانی پایدار کرد

مرتب سازی انتخابی (Selection Sort)

- ۷) آرایه n تایی را $n-1$ بار پیمایش میکند ، در هر پیمایش عنصر \max را پیدا میکند و با عنصر آخر تعویض میکند ، پس از هر پیمایش طول آرایه را یک واحد کم میکند (چون عناصر آخر آرایه مرتب است)

ویژگی ها :

درج

نامتعادل

پیچیدگی زمانی :

بهترین حالت : n^2

حالت متوسط :

بدترین حالت : n^2

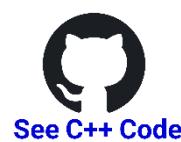
۸) الگوریتم و کد :

```

list  : array of items
n      : size of list

for i = 1 to n - 1
    max = i
    for j = i+1 to n
        if list[j] > list[max] then
            max = j;
        end if
    end for
    if min != i then
        swap list[max] and list[i]
    end if
end for
end procedure

```



مرتب‌سازی حبابی (Bubble Sort)

- ۱) آرایه n تایی را ۰-۱ بار پیمایش میکند، در هر پیمایش عنصر کنار هم را با هم مقایسه میکنداگر اولی از دومی بزرگتر بود جایشان را باهم تعویض میکند (در حالت صعودی)، پس از هر پیمایش طول آرایه را یک واحد کم میکند (چون عناصر آخر آرایه مرتب است)
- ۲) اگر بعد از یک بار پیمایش عنصری جا به جا نشود یعنی آرایه مرتب است و دیگر نیاز به بررسی مجدد ندارد (با یک flag می‌توان از بررسی مجدد جلو گیری کرد)

ویژگی‌ها :

درج

متداول

پیچیدگی زمانی :

n بهترین حالت :

n^2 حالت متوسط :

n^2 بدترین حالت :

۳) الگوریتم و کد :



See C# Code



See C++ Code

```

list : array of items
n    : size of list
flag = true
for i = 1 to n - 1

    flag = false

    for j = 1 to n-i-1

        if list[j] > list[j+1] then
            swap list[j] and list[j+1]
        flag = true
        end if

    end for

    if !flag then
        break;
    end if

end for
end procedure

```

- ۴) در بهترین حالت $n-1$ مقایسه انجام می‌شود ولی $\frac{n(n-1)}{2}$ جا به جایی برابر است

مرتب‌سازی درجی (Insertion Sort)

۱) آرایه n عنصری داریم ابتدا عنصر اول بعد دوم و ... را درج می‌کنیم، به این صورت که یک عنصر را (از ابتدای آرایه) یکی برشاشته و با عناصرهای قبلیش مقایسه می‌کنیم و جای آن را در میان عناصر قبلیش پیدا می‌کنیم

ویژگی‌ها:

درج

متداول

پیچیدگی زمانی:

بهترین حالت: n

حالت متوسط: n^2

بدترین حالت: n^2

۲) الگوریتم و کد:

```

list : array of items
n    : size of list

for i = 2 to n
    x = list[i]
    j = i-1

    while (list[j] > x && j > 0)
        list[j+1] = list[j]
        j--
    end for

    list[j+1] = y

end for

end procedure

```



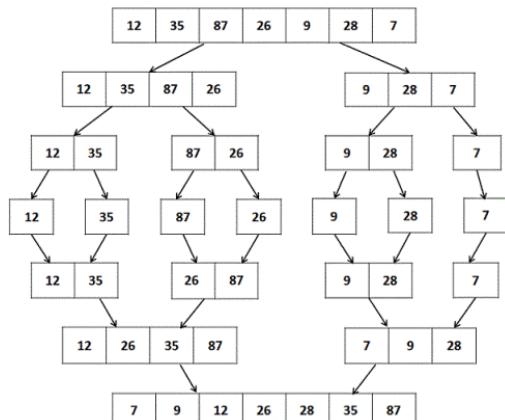
See C++ Code



See C# Code

۳) شرط $0 < \text{ز} < \infty$ را می‌توان با استفاده از یک sentinel حذف کرد، (اگر $\text{list}[0] = -\infty$)

مرتب سازی ادغامی (Merge Sort)



(۱) از نوع مرتب سازی های تقسیم و غلبه ای است و پیاده سازی آن به صورت بازگشتی است به همین جهت برای عناصر با تعداد کم مناسب نیست، در این الگوریتم هر بار آرایه n عنصری را به دو آرایه $\left\lfloor \frac{n}{2} \right\rfloor$ تقسیم می کند تا به آرایه یک عنصری برسد (که آرایه یک عنصری مرتب است)، سپس شروع میکند و آرایه های مرتب را باهم ادغام میکنم، (ادغام)(merge) : یعنی دو آرایه مرتب را به گونه ای باهم ادغام کنیم که آرایه حاصل نیز مرتب باشد) تا به کل آرایه برسیم :

(۲) ادغام :

۱. حداقل مقایسه برای ادغام آرایه های k, m عنصری برای است با $k+m-1$
- ✓ حداقل مقایسه برای ادغام آرایه های $\left\lfloor \frac{n}{2} \right\rfloor$ و $\left\lceil \frac{n}{2} \right\rceil$ عنصری برابر است با $n-1$
۲. حداقل مقایسه برای ادغام آرایه های k, m عنصری برای است با $\min\{k,m\}$
- ✓ حداقل مقایسه برای ادغام آرایه های $\left\lfloor \frac{n}{2} \right\rfloor$ و $\left\lceil \frac{n}{2} \right\rceil$ عنصری برابر است با $\left\lfloor \frac{n}{2} \right\rfloor$
۳. ادغام آرایه n عنصری و $\theta(n)$ می باشد

ویژگی ها :

درجا

متعدد

پیچیدگی زمانی :

بهترین حالت : $nlog(n)$

حالات متوسط : $nlog(n)$

بدترین حالت : n^2

(۳) الگوریتم و کد :

```
procedure merge( var a as array, var b as array )
var c as array
while ( a and b have elements )
  if ( a[0] > b[0] )
    add b[0] to the end of c
    remove b[0] from b
  else
    add a[0] to the end of c
    remove a[0] from a
  end if
end while
while ( a has elements )
  add a[0] to the end of c
  remove a[0] from a
end while
while ( b has elements )
  add b[0] to the end of c
  remove b[0] from b
end while
return c
end procedure
```

```
procedure mergesort( var a as array )
if ( n == 1 ) return a

var l1 as array = a[0] ... a[n/2]
var l2 as array = a[n/2+1] ... a[n]

l1 = mergesort( l1 )
l2 = mergesort( l2 )

return merge( l1, l2 )
end procedure
```

۶. چند نکته مهم پیرامون merge sort

$$(T(1) = 1, T(2) = 3) \rightarrow T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1 = 2n - 1$$

۷. تعداد فراخوانی در Merge sort :

$$(T(1) = 0, T(2) = 1) \rightarrow T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1 = n - 1$$

۸. چند بار فراخوانی می شود :

$$(T(1) = 0, T(2) = 1) \rightarrow T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n - 1$$

۹. حداکثر تعداد مقایسه های Merge :

۱۰. اگر n توانی از ۲ باشد حداکثر تعداد مقایسه ها Merge برابر است با $n \log n - n + 1$

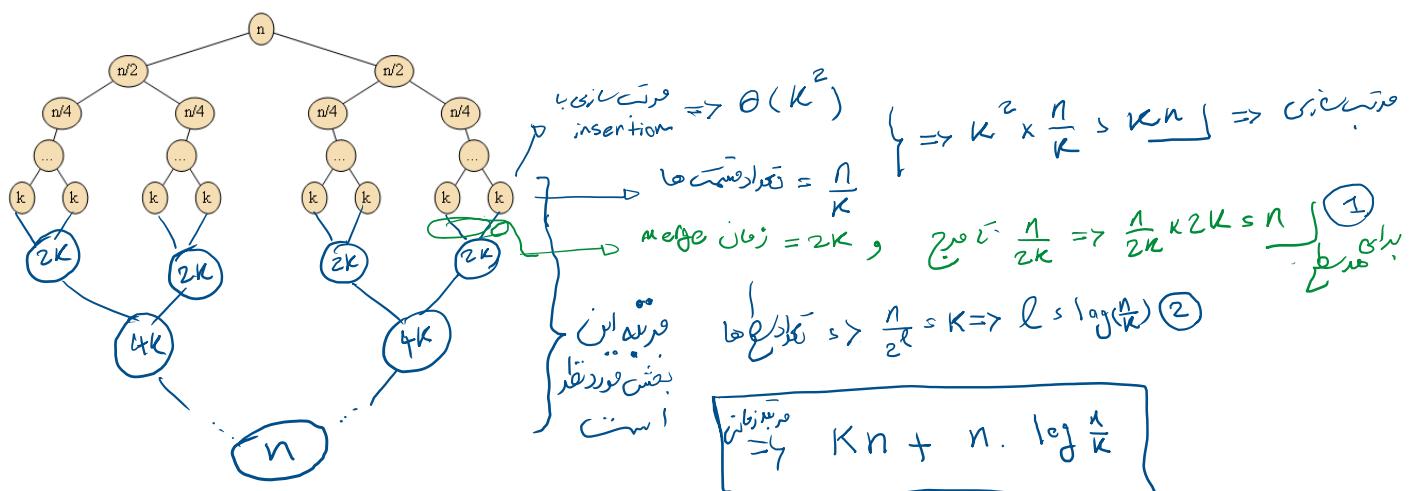
مثال : برای آرایه ۱۰ عنصری

الف) Merge sort چند بار فراخوانی می شود ؟

ب) Merge چند بار فراخوانی می شود ؟

ج) حداکثر تعداد مقایسه ها ؟

مثال : در مرتب سازی ادغامی فرض کن آرایه را آنقدر نصف کردیم که به لیست های K عنصری رسیدیم سپس این لیست های عنصری را با روش درجی مرتب کرده ایم و سپس شروع به ادغام کردیم، مرتبه زمانی را بر حسب k, n بیابیم.



مثال : ادغام K تا لیست مرتب n/k عنصری چقدر زمان می برد اگر

الف) ابتدا ۱ با ۲ سپس نتیجه اش با ۳ و ... ادغام کنیم ؟

ب) لیست ها را دو به دو ادغام کنیم ؟

ج) از هرم کمینه برای ادغام کمک بگیریم ؟

الف) برای ادغام لیست ۱ با ۲ که هر کدام $\frac{n}{k}$ عنصر دارند مقدار $\frac{2n}{k}$ زمان لازم است ادغام این دو لیست با لیست $\frac{n}{k}$ دیگر $\frac{3n}{k}$ زمان لازم است بنابراین داریم :

$$\frac{2n}{k} + \frac{3n}{k} + \frac{4n}{k} + \dots + n = \frac{n}{k}(2 + 3 + 4 + \dots + k) = \frac{n}{k}(\theta(k^2)) = \theta(nk)$$

ب) مانند مثال قبل رفتار کنید تا به $O(n \log k)$ برسید

ج) ک تا لیست داریم ، که عناصر سر هر لیست کمینه آن لیست است ، با این ها یک هرم کمینه بسازید ، سپس شروع کنید از ریشه به حذف کردن ، هر عنصری که حذف میکنیم در خروجی نگه می داریم و سپس از لیستی که آن عنصر به آن تعلق داشت یک عنصر جدید به هرم insert می کنیم و تکرار میکنیم . ساخت هرم کمینه زمان $\Theta(n \log k)$ است پس زمان $O(n \log k)$ می باشد.

➤ چند نکته مهم پیرامون مرتب سازی :

- ۱) اگر تعداد عناصر آرایه کم باشد یا آرایه از پیش مرتب شده باشد بهترین روش درجی است.
- ۲) اگر عناصر آرایه مساوی باشند روش های حبابی ، درجی و هرمی از مرتبه $\Theta(n^2)$ هستند.
- ۳) اگر آرایه از قبل مرتب باشد (در جهت الگوریتم) روش های حبابی و درجی $\Theta(n)$ هستند.
- ۴) روش های حبابی و درجی و سریع حداقل $\frac{n(n-1)}{2}$ مقایسه دارند ولی روش انتخابی دقیقاً $\frac{n(n-1)}{2}$ مقایسه دارد.
- ۵) اگر مقایسه هزینه نداشته باشد ولی جا به جایی هزینه زیادی داشته باشد بهترین روش انتخابی است چون تعداد جا به جایی در آن از همه روش ها کمتر است.
- ۶) اگر آرایه مرتب باشد روش های سریع و درختی بدترین حالت را دارند.

Shell Sort

۱) این روش از insertion sort گرفته شده در این روش از " اگر تعداد عناصر آرایه کم باشد یا آرایه تقریباً مرتب شده باشد بهترین روش درجی است ". استفاده کرد به این صورت که یک gap تعیین میکند مثل $\frac{n}{2}$ حال عناصری که باهم $\frac{n}{2}$ فاصله دارند را مرتب میکند سپس gap را کاهش می دهد مثل $\frac{n}{4}$ و این عمل را انجام می دهد تا $gap = 1$ شود در اینجا وقتی gap بزرگ است تعداد عناصر کم است و وقتی gap کاهش می یابد آرایه تقریباً مرتب است

ویژگی ها :

درج

متداول

پیچیدگی زمانی :

بهترین حالت : $O(n \log(n))$ یا $n^{1.5}$

حالت متوسط : $O(n \log(n))$ یا $n^{1.5}$

بدترین حالت : $O(n \log(n))$ یا $n^{1.5}$

۲) الگوریتم و کد :

```

procedure shellSort()
A : array of items

while interval < A.length /3 do:
    interval = interval * 3 + 1
end while

while interval > 0 do:

    for outer = interval; outer < A.length; outer ++ do:

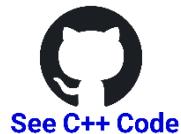
        valueToInsert = A[outer]
        inner = outer;

        while inner > interval -1 && A[inner - interval] >= valueToInsert do:
            A[inner] = A[inner - interval]
            inner = inner - interval
        end while
        A[inner] = valueToInsert

    end for
interval = (interval -1) /3;
end while

end procedure

```



[See C++ Code](#)



[See C# Code](#)

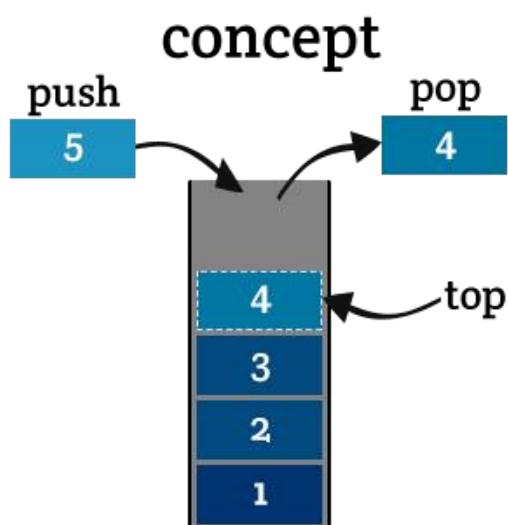
درخت تصمیم

۱. درختی دو دوبی که برای الگوریتم های مقایسه ای کشیده می شود و گره های داخلی مقایسه دو عنصر را نشان می دهند . یال چپ یعنی عنصر اول کوچکتر است و یال راست یعنی عنصر دوم کوچکتر است برگ درخت نتیجه یا خروجی را نشان میدهد.
۲. **نکته** : درخت دو دوبی با x برگ ارتفاعش حداقل $\lceil \log(x) \rceil$ است.
۳. برای مرتب سازی n عنصر تعداد برگ در درخت تصمیم $n!$ است زیرا هر جایگشت از عناصر میتواند یک خروجی باشد پس ارتفاع درخت تصمیم حداقل $\lceil \log(n!) \rceil$ است.
۴. بنابر این در بدترین حالت برای مرتب کردن n عدد حداقل $\lceil \log(n!) \rceil$ مقایسه لازم است.
۵. مرتب سازی مقایسه ای در بدترین حالت از مرتبه $\Omega(n \log n)$ است و همچنین این نتیجه برای حالت متوسط نیز درست است.
۶. ادغام k تا لیست مرتب $\frac{n!}{(\frac{n}{k})!^k}$ عنصری تعداد حالاتش در نتیجه تعداد برگ درخت تصمیم است که برابر است با
۷. هر روشی که بخواهد k تا لیست مرتب $\frac{n!}{(\frac{n}{k})!^k}$ عنصری را ادغام کند و یک لیست n تایی تحویل دهد در بدترین حال از مرتبه $\Omega\left(\log\left(\frac{n!}{(\frac{n}{k})!^k}\right)\right) = \Omega(n \log k)$

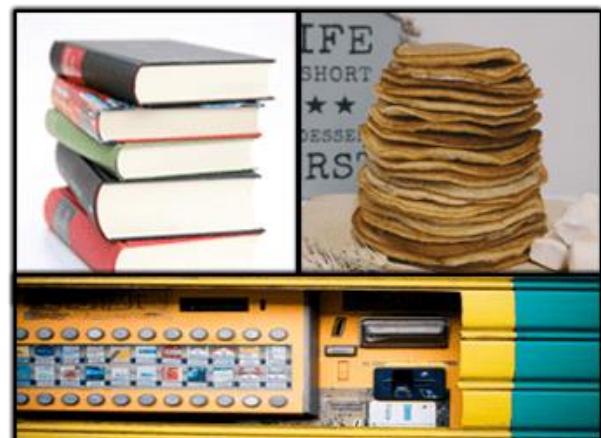
پشته

مجموعه از داده که عملیات درج و حذف روشون انجام میشه ، در این مجموعه آخرین عنصری که وارد می شود ، اولین عنصری است که خارج می شود. کابرد های زیاده دارد ، مثلا فراختانی تابع ، Backtracking و

Stack



real life



last-in-first-out (LIFO)

- x را بالای پشته قرار می دهد.
- عنصر بالای پشته را از آن خارج می کند.
- همواره به بالای پشته اشاره می کند.

مثال : فرض کنید اعداد ۱ تا ۵ را به ترتیب داخل Stack می کنیم و هر وقت بخواهیم Pop می کنیم و خروجی را چاپ می کنیم کدام خروجی قابل تولید نیست ؟

۳۴۱۵۲(د)

۴۳۵۲۱(ج)

۳۴۵۲۱(ب)

۳۲۴۱۵(الف)

بررسی گزینه ها : (از چپ به راست بخوانید ، Pop عنصر بالای پشته را از آن خارج میکند جهت تفهیم مقدار داخل پرانتز نوشته شده است)

(الف) Push(۱) Push(۲) Push(۳) Push(۴) Push(۵) Push(۶) Push(۷) Push(۸) Push(۹) Push(۱۰) Push(۱۱) Push(۱۲)

(ب) Push(۱) Push(۲) Push(۳) Push(۴) Push(۵) Push(۶) Push(۷) Push(۸) Push(۹) Push(۱۰) Push(۱۱) Push(۱۲)

(ج) Push(۱) Push(۲) Push(۳) Push(۴) Push(۵) Push(۶) Push(۷) Push(۸) Push(۹) Push(۱۰) Push(۱۱) Push(۱۲)

(ج) (۲) Push(۱) Push(۲) Push(۳) Push(۴) Push(۵) Push(۶) Push(۷) Push(۸) Push(۹) Push(۱۰) Push(۱۱) Push(۱۲)

از قبل وارد شده است و اول باید ۲ خارج شود.

می دانیم با n عدد ! n جایگشت می توان ساخت ولی تمامی این جایشگت ها با Stack قابل تولید نیست ، تعداد جایگشت

$$C_n = \frac{1}{n+1} \binom{2n}{n}$$

پیاده سازی با آرایه :

ساختار کلاس :

```
#define MAXSIZE 1000      تعداد خانه های پشته
template <class T>
class Stack {
    int top;            اشاره گر به آخرین عنصر داخل استک
public:
    T a[MAXSIZE];     بافر استک
    Stack();           سازنده کلاس
    bool push(T x);   افزودن عنصر به استک
    int pop();         حذف عنصر بالایی از استک
    int peek();        مشاهده عنصر بالا بدون حذف شدن
    bool isEmpty();   بررسی خالی بودن استک
    bool isfull();    بررسی پر بودن استک
};
```

در این پیاده سازی top به آخرین خانه پُر اشاره میکند :

سازنده کلاس :

```
template <class T>
Stack<T>::Stack()
{
    top = -1;
}
```

: کردن Push

```
template <class T>
bool Stack<T>::push(T x)
{
    if (isfull()) {
        throw ("Stack Overflow");
    }
    else {
        a[++top] = x;
        return true;
    }
}
```

: کردن Pop

```
template <class T>
int Stack<T>::pop()
{
    if (isEmpty()) {
        throw ("Stack Underflow");
    }
    else {
        int x = a[top--];
        return x;
    }
}
```

مشاهده عنصر بالابی(Peek):

```
template <class T>
int Stack<T>::peek()
{
    if (isEmpty()) {
        throw ("Stack is Empty");
    }
    else {
        int x = a[top];
        return x;
    }
}
```

بررسی خالی بودن :

```
template <class T>
bool Stack<T>::isEmpty()
{
    return top == -1;
}
```

بررسی پر بودن :

```
template <class T>
bool Stack<T>::isfull() {
    return top == MAXSIZE;
}
```



➤ پیاده سازی با لیست پیوندی :

[برای مشاهده پیاده سازی لیست پیوندی به گیت هاب مراجعه کنید.](#)

[برای مشاهده استک به وسیله لیست پیوندی به گیت هاب مراجعه کنید.](#)

فرم های ریاضی

$$\left\{ \begin{array}{l} Infix : a + b * c / (a + b) \\ Postfix (Polish Notation(NP)) : a b c * a b + / + \\ PreFix (Revers Polish Notation(RNP)) : + a / * b c + a b \end{array} \right.$$

-۱ infix : در این بخش مشکلی که وجود دارد این است که علیت ها ابهام دارد و باید از پرانتز استفاده کرد

- شکل کلی آن : عملوند عملگر عملوند

-۲ : Postfix

- شکل کلی آن : عملگر عملوند عملوند

-۳ : Prefix

- شکل کلی آن : عملوند عملوند عملگر

تبديلات :

➤ تبديل postfix به infix با استفاده از پانتزگذاري :

-۱ عبارت $(a + ((b * c) / (a + b)))$ را بنا به الويت پرانتز گذاري ميكنيم :

-۲ سپس عملگرها را (operator) به بعد از پرانتز های مربوطه منتقل ميكنيم

$(a((bc) * (ab)) /) +$

-۳ سپس پرانتز هارا حذف ميكنيم

$abc * ab + / +$

➤ تبديل prefix به infix با استفاده از پانتزگذاري :

-۱ عبارت $(a + ((b * c) / (a + b)))$ را بنا به الويت پرانتز گذاري ميكنيم :

-۲ سپس عملگرها را (operator) به قبل از پرانتز های مربوطه منتقل ميكنيم

$+ (a / (* (bc) + (ab)))$

-۳ سپس پرانتز هارا حذف ميكنيم

$+ a / * bc + ab$

➤ تبديل postfix به infix با استفاده از پانتزگذاري :

-۱ عبارت $abc * ab + / +$ را بنا به صورت عملگر عملوند عملوند پرانتز گذاري می کنیم (+)

-۲ سپس عملگرها را (operator) به میان دو عملوند مربوطه منتقل ميكنيم

$(a + ((b * c) / (a + b)))$

-۳ سپس پرانتز هارا حذف ميكنيم (دليل حذف نشدن پرانتز a+b اين است که قبل از آن تقسيم است)

$a + b * c / (a + b)$

➤ تبديل prefix به postfix با استفاده از پانتزگذاري :

-۱ عبارت $+ abc * ab + / +$ را بنا به صورت عملگر عملوند عملوند پرانتز گذاري می کنیم (+)

-۲ سپس عملگرها را (operator) به قبل از پرانتز های مربوطه منتقل ميكنيم

$+ (a / (* (bc) + (ab)))$

-۳ سپس پرانتز هارا حذف ميكنيم

$+ a * bc + ab$

➤ تبديل infix به prefix با استفاده از پانتزگذاري :

- عبارت $(+a(/(*bc)(+ab))$ را بنا به صورت عملوند عملگر پرانتز گذاری می کنیم

- سپس عملگر ها را (operator) به میان دو عملوند مربوطه منتقل میکنیم

$$(a + ((b * c) / (a + b)))$$

- سپس پرانتز هارا حذف میکنیم (دلیل حذف نشدن پرانتز b این است که قبل از آن تقسیم است)

$$a + b * c / (a + b)$$

➢ تبدیل postfix به prefix با استفاده از پانترگذاری :

- عبارت $(+a(/(*bc)(+ab))$ را بنا به صورت عملوند عملگر پرانتز گذاری می کنیم

- سپس عملگر ها را (operator) به بعد از پرانتز های مربوطه منتقل میکنیم

$$(a((bc) * (ab) +) +)$$

- سپس پرانتز هارا حذف میکنیم

$$(abc * ab +) +$$

➢ تبدیل postfix به infix با استفاده از استک :

➢ به دو استک نیاز است استک عملگر و استک عملوند به ابتدا و انتها عبارت $a + b * c / (a + b)$ پرانتز اضافه میکنیم

یعنی $(a + b * c / (a + b))$ سپس شروع می کنیم و داخل استک push میکنیم تا زمانی که استک عملگر ها به عملگری برسد که الوبیت آن از عملگر های موجود در استک تا قبل از اولین پرانتز باز کمتر یا مساوی باشد وقتی به این عملگر یا پرانتز بسته رسیدیم تا پرانتز باز قبلی باید استک را تخلیه کنیم نحوره تخلیه استک به صورت عملوند عملوند عملگر است توجه کنید که عملوند پایین تر باید اول بیاید.

قدم اول

c	*
b	+
a	(
عملوند	عملگر

وضعیت فعلی عبارت :

$((a + b))$

به تقسیم رسیدیم که الوبیش با ضرب مساوی است پس تا پرانتز باز تکلیف را مشخص می کنیم .

قدم دوم

$bc *$	+
a	(
عملوند	عملگر

دو عملوند و یک عملگر pop می کنیم تا به اولین پرانتز باز برسیم

قدم سوم

	+
b	(
a	/
$bc *$	+
a	(
عملوند	عملگر

وضعیت فعلی عبارت :

)

قدم چهارم

$ab +$	/
$bc *$	+
a	(
عملوند	عملگر

دو عملوند و یک عملگر pop می کنیم تا به اولین پرانتز باز برسیم

قدم پنجم

$ab +$	/
$bc *$	+
a	(
عملوند	عملگر

وضعیت فعلی عبارت :

)

قدم پنجم -۱

$bc * ab +/$	+
a	(
عملوند	عملگر

دو عملوند و یک عملگر pop می کنیم تا به اولین پرانتز باز برسیم

قدم پنجم -۲

$abc * ab +/+$	(
عملوند	عملگر

دو عملوند و یک عملگر pop می کنیم تا به اولین پرانتز باز برسیم

قدم پنجم -۳

عملوند	عملگر

استک عملگر ها خالی شد و به جواب نهایی رسیدیم :

$$abc * ab +/+$$

به پانتر بسته رسیدیم

➢ تبدیل infix به prefix با استفاده از استک :

▶ به دو استک نیاز است استک عملگر و استک عملوند به ابتدا و انتهای عبارت $a + b * c / (a + b)$ پرانتز اضافه می کنیم یعنی $(a + b * c / (a + b))$ سپس شروع می کنیم و داخل استک push می کنیم تا زمانی که استک عملگر ها به عملگری برسد که الوبیت آن از عملگر های موجود در استک تا قبل از اولین پرانتز باز **کمتر یا مساوی** باشد وقتی به این عملگر یا پرانتز بسته رسیدیم تا پرانتز باز قبلی باید استک را تخلیه کنیم نحوره تخلیه استک به صورت عملگر عملوند عملوند است ، توجه کنید که عملوند پایین تر باید اول بیاید .

قدم اول

c	*
b	+
a	(
عملوند	عملگر

وضعیت فعلی عبارت :

 $/(a + b))$

به تقسیم رسیدیم که الوبیش با ضرب مساوی است پس تا پرانتز باز تکلیف را مشخص می کنیم .

قدم پنجم

+ab	/
* bc	+
a	(
عملوند	عملگر

وضعیت فعلی عبارت :

)

به پانتر بسته رسیدیم

* bc	+
a	(
عملوند	عملگر

دو عملوند و یک عملگر pop می کنیم تا به اولین پرانتز باز برسیم

قدم پنجم

+a/* bc + ab	
عملوند	عملگر

پاسخ نهایی :

 $+a/* bc + ab$

قدم سوم

	+
b	(
a	/
* bc	+
a	(
عملوند	عملگر

وضعیت فعلی عبارت :

))

به پانتر بسته رسیدیم

قدم چهارم

+ab	/
* bc	+
a	(
عملوند	عملگر

دو عملوند و یک عملگر pop می کنیم تا به اولین پرانتز باز برسیم

)

)

- ▶ تبدیل infix به postfix با استفاده از استک : مشابه حالت های قبل اگر قبل از جمع یا تفریق ضرب یا تقسیم بایشد آن جمع یا تفریق باید داخل پرانتز قرار گیرد
- ▶ تبدیل infix به prefix با استفاده از استک : مشابه حالت های قبل اگر قبل از جمع یا تفریق ضرب یا تقسیم بایشد آن جمع یا تفریق باید داخل پرانتز قرار گیرد.
- ▶ تبدیل postfix به prefix با استفاده از استک : مشابه حالت های قبل اگر قبل از جمع یا تفریق ضرب یا تقسیم بایشد آن جمع یا تفریق باید داخل پرانتز قرار گیرد.
- ▶ تبدیل postfix به infix با استفاده از استک : مشابه حالت های قبل اگر قبل از جمع یا تفریق ضرب یا تقسیم بایشد آن جمع یا تفریق باید داخل پرانتز قرار گیرد.

ساخت چند پشته در یک آرایه

برای ساخت ۲ پشته در یک آرایه بهتر آن است که یکی از پشته ها از ابتدا و دیگری از انتهای آرایه شروع به پرسد کند و زمانی که top اولی به m دومی رسید پشته ها را پُر اعلام کنیم(اگر آرایه را نصف کنیم ممکن است آرایه جای خالی داشته باشد ولی یکی از پشته ها پرسد) ، اما برای بیش از دو آرایه این کار بسیار دشوار است .

برای ساخت m تا پشته در آرایه n تایی داریم :

۱	۲	۳							n
---	---	---	--	--	--	--	--	--	-----

اندازه هر پشته $\left\lfloor \frac{n}{m} \right\rfloor$ است البته ممکن است پشته آخر کمی بیشتر شود

برای هر آرایه یک top و یک bottom داریم ، البته bottom یک عدد بیشتر است که به انتهای آرایه اشاره می کند.

$Top[1...m], bottom[1...m+1]$

مقدار اولیه را داریم : $top(i) = bottom(i) = (i - 1) * \left\lfloor \frac{n}{m} \right\rfloor$

بنابراین شروع پشته i ام : $Start(i) = (i - 1) * \left\lfloor \frac{n}{m} \right\rfloor + 1$

برای push کردن داریم :

```
If(top[i] != bottom[i+1])
{
    Top[i]++;
    Array[Top[i]] = x;
}
```

برای pop کردن داریم :

```
If(top[i] != bottom[i])
{
    X = Array[Top[i]];
    Array[Top[i]]--;
}
```

The Maze Problem

Source			
			Dest.

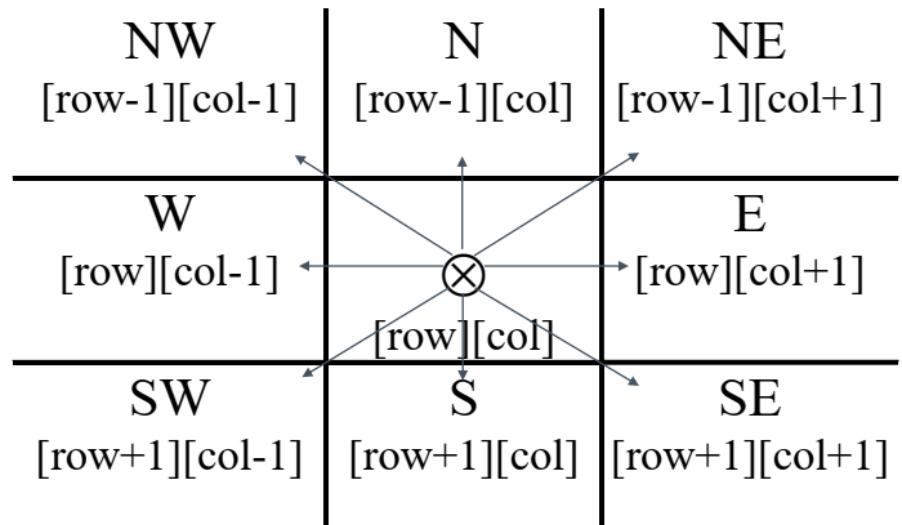
(میز) به راه‌های تو در تو گفته می‌شود، که باید از یک مکان وارد و از مکان دیگر از آن خارج شد. به عبارت دقیق‌تر، Maze یک ناحیه شبکه‌ای شکل دو بعدی است که شامل سلوول‌هایی می‌باشد. یک Maze می‌تواند شامل موانع مختلف و با هر تعدادی باشد. پیچیدگی Maze بسته به تعداد سلوول‌ها، موانع، راهرو‌ها و بن بست‌ها و فاصله بین سلوول شروع تا پایان، متفاوت می‌باشد.

هدف در این مسئله اتخاذ ترتیبی از تصمیمات به منظور رسیدن به حالت هدف از حالت شروع می‌باشد.

ماتریس باینری زیر نمایشی از maze بالا می‌باشد.

```
{1, 0, 0, 0}
{1, 1, 0, 1}
{0, 1, 0, 0}
{1, 1, 1, 1}
```

برای مدل سازی میز از ۰ به عنوان مسیر و از ۱ و دیواره‌های ماتریس به عنوان دیوار استفاده می‌کنیم، به طور کلی یک خانه ۸ انتخاب برای حرکت دارد:



در جهت یابی بنا بر محل خروج ما به جهت‌ها الوبیت می‌دهیم (در صورتی که دوست دارید این کار را انجام دهید).

فرض کنید مسئله maze برای ماتریس زیر تعریف شده است:

```
{ 0, 1, 0, 0, 1 },
{ 0, 0, 0, 1, 1 },
{ 1, 0, 1, 0, 0 },
{ 0, 0, 0, 0, 0 }
```

موس ما از خانه (۰,۰) شروع به حرکت می‌کند، ابتدا چک می‌کنیم که می‌تواند شمال حرکت کند یعنی یکی از شماره ریف آن کم شود یا نه که امکان ندارد چون ردیف ۱- می‌شود، سپس شمار شرقی و ... را چک می‌کنیم تا به جنوب شرقی مرسمیم که صفر است پس موس ما از (۰,۰) به خانه (۱,۱) می‌آید مجدد از اول شروع می‌کنیم و جهت هارا چک می‌کنیم، یک آرایه کمکی تعریف می‌کنیم و قتی وارد یک خانه می‌شویم خانه متناظر را در آرایه کمکی یک می‌کنیم که متوجه بشیم قبلًا در این خانه بود

ایم، تا مجدد به آن خانه نیایم، اگر به بن بست رسیدیم یک خانه به عقب بر می‌گیریم و سایر جهت‌های آن خانه را چک می‌کنیم باز اگر جهتی برای حرکت وجود نداشت به عقب باز می‌گردیم، اگر به جایی برسیم که همه طرف‌ها فقل باشد می‌توان نتیجه گرفت مسئله قابل حل نیست، همچنین در آرایه $m^*p/2$ خانه استک نیاز داریم چرا که در بدترین وضعیت زمانی رخ میدهم که مسیر مارپیچ باشد و ما مجبور باشیم نصف خانه‌هارا مشاهده کنیم.(اعداد قرمز خانه‌هایی است که موش ملاقات می‌کند).

پیاده‌سازی مسئله Maze

ساختر مسیر‌ها :

```
#include <cstring>
#include <iostream>
#include <stack>
using namespace std;
enum Direction
{
    N,
    NE,
    E,
    SE,
    S,
    SW,
    W,
    NW,
    None
};

struct Driver
{
    int X;
    int Y;
    Direction direction;
    Driver(int i, int j) : X(i), Y(j), direction(Direction::N) {}
    void NextDirction()
    {
        if(direction != Direction::None)
            direction = (Direction)((int)direction+1);
    }
};
```

ساختر حرکت کننده :

```
class Maze
{
public:
    int MapRowSize;
    int MapColumnSize;
    int DestinationX;
    int DestinationY;
    int** MazeMap;
    bool** visited;
    int startX;
    int startY;
    Maze::Maze(int** map, int MapRowSize, int MapColumnSize, int DestinationX, int DestinationY, int startX, int startY);
    inline bool Maze::isSafe(int x, int y);
    bool Maze::isReachable();
};
```

سازنده کلاس :

```
Maze::Maze(int** map, int MapRowSize, int MapColumnSize, int DestinationX, int DestinationY, int startX, int StartY)
    :MapRowSize(MapRowSize), MapColumnSize(MapColumnSize), DestinationX(DestinationX), DestinationY(DestinationY), startX(startX), StartY(StartY)
{
    MazeMap = new int*[MapRowSize];
    visited = new bool*[MapRowSize];
    for (size_t i = 0; i < MapRowSize; i++)
    {
        MazeMap[i] = new int[MapColumnSize];
        visited[i] = new bool[MapColumnSize];
        for (size_t j = 0; j < MapColumnSize; j++)
        {
            MazeMap[i][j] = map[i][j];
            visited[i][j] = false;
        }
    }
}
bool Maze::isSafe(int x, int y)
{
    return (x >= 0 && x <= MapColumnSize && y >= 0 && y <= MapRowSize && MazeMap[x][y] == 0 && !visited[x][y]);
}
```

تابع بررسی کننده درستی مسیر :

```
bool Maze::isSafe(int x, int y)
{
    return (x >= 0 && x <= MapColumnSize && y >= 0 && y <= MapRowSize && MazeMap[x][y] == 0 && !visited[x][y]);
}
```

تابع حل مسئله :

```
bool Maze::isReachable()
{
    stack<Driver> s;

    Driver temp(startX, StartY);
    s.push(temp);

    while (!s.empty()) {
        temp = s.top();
        Direction currentDirection = temp.direction;
        int currentX = temp.X;
        int currentY = temp.Y;
        temp.NextDirction();
        s.pop();
        s.push(temp);

        // If we reach the Food coordinates
        // return true
        if (currentX == DestinationX && currentY == DestinationY) {
            return true;
        }

        switch (currentDirection)
        {
        case N:
            if (isSafe(currentX - 1, currentY)) {
                Driver temp1(currentX - 1, currentY);
                visited[currentX-1][currentY] = true;
                s.push(temp1);
            }
            break;
        case NE:
            if (isSafe(currentX - 1, currentY+1)) {
                Driver temp1(currentX - 1, currentY + 1);
                visited[currentX-1][currentY+1] = true;
                s.push(temp1);
            }
            break;
        case E:
            if (isSafe(currentX, currentY+1)) {
                Driver temp1(currentX, currentY + 1);
                visited[currentX][currentY+1] = true;
                s.push(temp1);
            }
            break;
        case SE:
            if (isSafe(currentX + 1, currentY+1)) {
                Driver temp1(currentX + 1, currentY + 1);
                visited[currentX+1][currentY+1] = true;
                s.push(temp1);
            }
            break;
        case S:
            if (isSafe(currentX + 1, currentY)) {
                Driver temp1(currentX + 1, currentY);
                visited[currentX+1][currentY] = true;
                s.push(temp1);
            }
            break;
        case SW:
            if (isSafe(currentX + 1, currentY-1)) {
                Driver temp1(currentX + 1, currentY - 1);
                visited[currentX+1][currentY-1] = true;
                s.push(temp1);
            }
            break;
        case W:
            if (isSafe(currentX, currentY-1)) {
                Driver temp1(currentX, currentY - 1);
                visited[currentX][currentY-1] = true;
                s.push(temp1);
            }
            break;
        }
    }
}
```

```

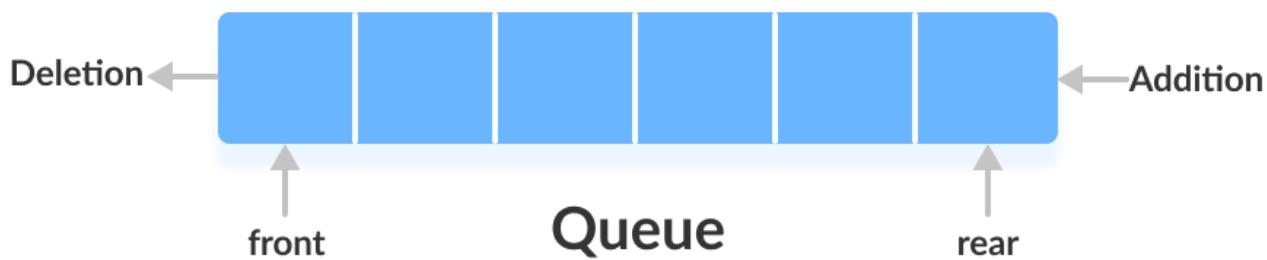
        visited[currentX - 1][currentY + 1] = true;
        s.push(temp1);
    }
    break;
case E:
    if (isSafe(currentX , currentY + 1)) {
        Driver temp1(currentX , currentY + 1);
        visited[currentX][currentY + 1] = true;
        s.push(temp1);
    }
    break;
case SE:
    if (isSafe(currentX+1, currentY + 1)) {
        Driver temp1(currentX+1, currentY + 1);
        visited[currentX+1][currentY + 1] = true;
        s.push(temp1);
    }
    break;
case S:
    if (isSafe(currentX + 1, currentY)) {
        Driver temp1(currentX + 1, currentY);
        visited[currentX + 1][currentY] = true;
        s.push(temp1);
    }
    break;
case SW:
    if (isSafe(currentX + 1, currentY - 1)) {
        Driver temp1(currentX + 1, currentY - 1);
        visited[currentX + 1][currentY - 1] = true;
        s.push(temp1);
    }
    break;
case W:
    if (isSafe(currentX , currentY - 1)) {
        Driver temp1(currentX , currentY - 1);
        visited[currentX ][currentY - 1] = true;
        s.push(temp1);
    }
    break;
case NW:
    if (isSafe(currentX - 1, currentY - 1)) {
        Driver temp1(currentX - 1, currentY - 1);
        visited[currentX - 1][currentY - 1] = true;
        s.push(temp1);
    }
    break;
case None:
    // If none of the direction can take the rat to the Food, retract back
    // to the path where the rat came from.
    visited[currentX][currentY] = true;
    s.pop();
    break;
default:
    throw exception();
    break;
}
}

// If the stack is empty and
// no path is found return false.
return false;
}

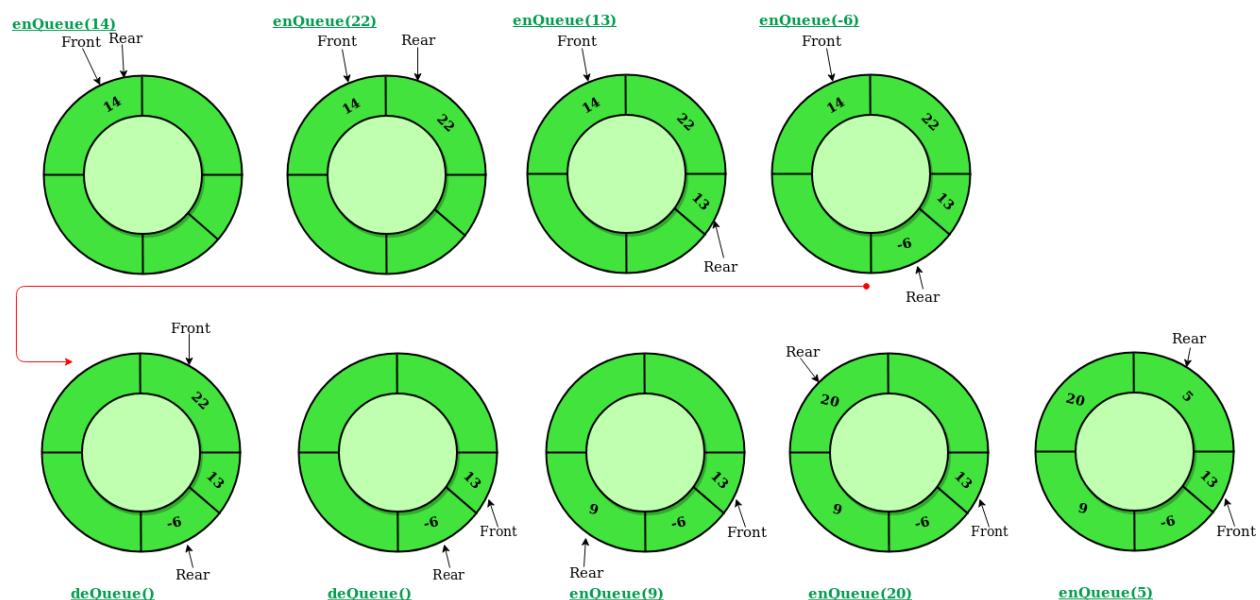
```

صف

- مجموعه از داده که عملیات درج و حذف روشون انجام میشه ، در این مجموعه اولین عنصری که وارد می شود . به صورت FIFO می باشد.
- صفات ساده :



➤ صفات حلقوی :



➤ به ابتدای صف اشاره می کند.

► به انتهای صف اشاره میکند. Rear

- پیاده سازی با آرایه(در آرایه بهینه تر آن است که صف به صورت حلقوی پیاده سازی شود از این رو صف حلقوی
- پیاده سازی شده است) :
- پیاده سازی اول صف حلقوی :
- در این پیاده سازی برای کنترل پر و خالی بودن صف از یک متغیر کمکی استفاده میکنیم اگر مقدار آن صفر باشد یعنی صف خالی است و اگر مقدار آن با اندازه صف برابر باشد یعنی صف پر است.

کلاس صف :

```
class CircularQueue
{
private:
    int *Buffer;
    int front = 0;
    int rear = 0;
    int FullSize = 0;
    int size;
public:
    CircularQueue(int size);
    ~CircularQueue();
    void addq(int obj);
    int delq();
    bool IsFull();
    bool IsEmpty();
    void Move(int& tmp);
};
```

سازنده و مخرب کلاس :

```
CircularQueue::CircularQueue(int size):size(size)
{
    Buffer = new int[size];
}
CircularQueue::~CircularQueue()
{
    delete Buffer;
}
```

حرکت دادن rear و front به جلو ، اگر به انتهای برسد باید بر گردد از اول :

```
void CircularQueue::Move(int& tmp)
{
    if(tmp == size)
        tmp = 0;
    else
        tmp++;
}
```

افزودن عنصر به صف :

```
void CircularQueue::addq(int obj)
{
    if(!IsFull())
    {
        Buffer[rear] = obj;
        Move(rear);
        FullSize++;
    }
    else throw exception();
}
```

حذف عنصر ابتدای صف :

```
int CircularQueue::delq()
{
    if(!IsEmpty())
    {
        int res = Buffer[front];
        Move(front);
        FullSize--;
        return res;
    }
    else throw exception();
}
```

بررسی پر بودن و خالی بودن :

```
bool CircularQueue::IsFull()
{
    return FullSize == size;
}
bool CircularQueue::IsEmpty()
{
    return FullSize == 0;
}
```

➤ پیاده سازی دوم صف حلقوی :

- در صف حلقوی اگر $front < rear$ به یک جا اشاره کننده صف خالی است و زمانی که $front = rear$ یک خانه فاصله داشته باشد صف پر است دلیلیش این است اگر $front = rear$ رویه هم منطبق شوند صف خالی در نظر گرفته میشود درحالی که پر است ، در این پیاده سازی یک خانه بیشتر نیاز داریم ولی متغیر کمکی دیگر نداریم .
- برای تعداد خانه های پر را بدست آوریم داریم :

$$if (front < rear) \text{ then } full = rear - front$$

$$if (front \geq rear) \text{ then } full = size - (front - rear)$$

➤ هم چنین می دانیم که روابط زیر برقرار است :

$$if a < b \text{ then } a \bmod b = a$$

$$n + 1 \bmod n = 1 \quad \text{و} \quad n + 2 \bmod n = 2 \quad \text{و} \quad \dots$$

$$n - 1 \bmod n = n - 1 \quad \text{و} \quad n - 2 \bmod n = n - 2 \quad \text{و} \quad \dots$$

➤ اگر روابط گفته شده برای بدست آوردن $full$ را با توجه به نکات بالا بازنویسی کنیم داریم :

$$if (front < rear) \text{ then } full = rear - front$$

$$if (front \geq rear) \text{ then } full = size - (front - rear) = size + (rear - front)$$

➤ در روابط بالا جایگذاری میکنیم :

$$full = (size + (rear - front)) \bmod size$$

➤ اگر $size > size + (rear - front)$ منفی می شود که باعث می شود $(rear - front)$ از

از $size + (rear - front)$ به باقی مانده بگیریم می شود

➤ اگر $size + (rear - front) > size$ مقدار $rear - front$ از $size + (rear - front)$ مثبت است و

است یعنی اگر از $size + (rear - front)$ باقی مانده بگیریم $(rear - front)$ می شود .

➤ کلاس صف :

```
class CircularQueue
{
private:
    int *Buffer;
    int front = 0;
    int rear = 0;
    int size;
public:
    CircularQueue(int size);
    ~CircularQueue();
    void addq(int obj);
    int delq();
    bool IsFull();
    bool IsEmpty();
    void Move(int& tmp);
};
```

سازنده و مخرب کلاس :

```
CircularQueue::CircularQueue(int size):size(size+1)
{
    Buffer = new int[size+1];
}
CircularQueue::~CircularQueue()
{
    delete Buffer;
}
```

: front و rear حرکت

```
void CircularQueue::Move(int& tmp)
{
    if(tmp == size)
        tmp = 0;
    else
        tmp++;
}
```

افزودن به صف :

```
void CircularQueue::addq(int obj)
{
    if(!IsFull())
    {
        Buffer[rear] = obj;
        Move(rear);
    }
    else throw exception();
}
```

حذف کردن از صف :

```
int CircularQueue::delq()
{
    if(!IsEmpty())
    {
        int res = Buffer[front];
        Move(front);
        return res;
    }
    else throw exception();
}
```

بررسی پر بودن صف :

```
bool CircularQueue::IsFull()
{
    return size+(rear-front) == size-1;
}
```

بررسی خالی بودن صف :

```
bool CircularQueue::IsEmpty()
{
    return rear == front;
}
```

پیاده سازی با لیست پیوندی :

هر صف node را کلاس صفت :

```
struct QNode {
    int value;
    QNode* next ;
    QNode(int value) : value(value), next(nullptr){ }
};
```

کلاس صفت :

```
class Queue {
public:
    QNode *front;
    QNode *rear;
    Queue()
    {
        front = rear = nullptr;
    }
    bool IsFull();
    void addq(int x)
    {
        QNode* newNode = new QNode(x);
        if (rear == nullptr) // first Node
            front = rear = newNode;
        else
        {
            rear->next = newNode;
            rear = newNode;
        }
    }
    int deQueue()
    {
        if (front != nullptr)
        {
            QNode* temp = front;
            front = front->next;

            if (front == nullptr)
                rear = nullptr;

            int res = temp->value;
            delete (temp);
            return res;
        }
        else throw exception();
    }
}
```

Double-ended Queue

در این نوع از صف ها هم میتوان به ابتدای صف افزود هم به انتهای صف عناصری که الوبت بیشتر برای رسیدگی دارند به ابتدای صف افزوده می شوند.

کاربردهای صف : پردازش برنامه ها در سیستم عامل ، پرینت مطالب ارسال شده به پرینتر ، پکت های داده در ارسال درون شبکه و

➢ میدانیم مرتبه جست و جو خطی : $O(n)$ و مرتبه جست و جو دودویی : $O(\log(n))$

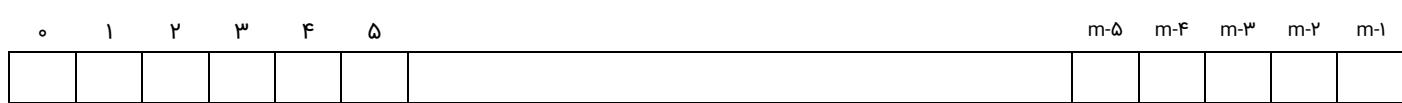
درهم سازی (Hashing)

- هدف از درهم سازی (hashing) این است که جست و جو با مرتبه $O(1)$ باشد ، به این صورت که به هر عنصر یک کلید متمایز بدهیم . درواقع هر عنصر یک کلید دارد و یک داده وابسته (satellite data).
- مثلا برای دانشجو ، شماره دانشجویی کلید و سایر اطلاعات دانشجو satellite data می باشد.

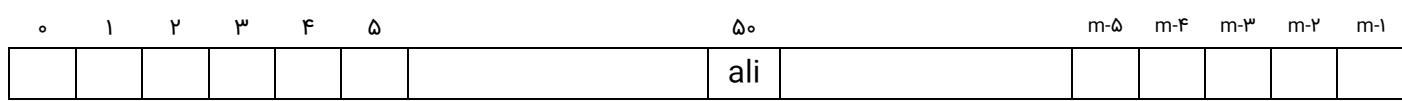
Direct Hashing

درهم سازی مستقیم شامل یک آرایه است (Hash Table) که تعداد خانه های آن به اندازه تعداد کلید هاست.

اگر کلید ها به صورت : $u = \{0, 1, 2, \dots, m - 1\}$ باشد برای Hash Table به یک آرایه m عنصری نیاز است .



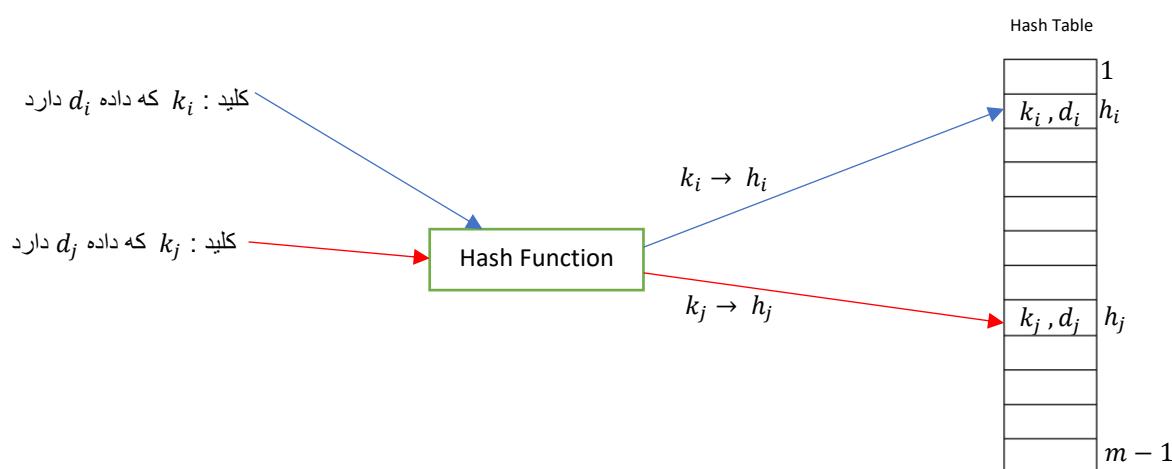
مثالا اگر کلید ali برابر باشد با 50 آنگاه علی را در خانه 50 می گذاریم . در واقع اندیس های آرایه کلید هستند.



این روش زمانی خوب است که دامنه زیاد بزرگ نباشد فرض کنید که مرجع کلید ها از 0 تا یک میلیون باشد ولی ما 5 دانشجو داشته باشیم آن وقت باید آرایه یک میلیون عنصری درست کنیم درحالی که فقط از 5 عنصرش استفاده می کنیم که باعث اتلاف زیاد حافظه می شود اما مزیتی که دارد این است که مرتبه جست و جو دقیقاً $O(1)$ است چون با داشتن کلید مستقیم از آرایه آن را بر میداریم.

نکته : در hashing ما بر اساس کلید جست و جو می کنیم.

برای رفع مشکل در هم سازی مستقیم : یکتابع تعریف میکنیم و کلید عناصر را به تابع می دهیم که تابع یک آدرس در یک فضای محدود تولید کند. مثلاً فرض کنید ما یک میلیون کلید داریم (دنیا کلید ها وسیع است) اما تعداد داده ها محدود است ، ارایه ای با تعداد محدود (کمتر از کلید ها) تعریف میکنیم و تابعی که کلید میگیرد و آدرسی تولید میکند که آن آدرس ، اندیس آرایه آرایه است ، در این صورت ما کلید های زیاد را در حافظه ای محدود ذخیره سازی کردیم.



برای جستجو هم مثلا کلید k_i را به تابع می دهیم تا آدرس (اندیس) h_i را تولید کند و می رویم از آرایه بر می داریم.

مشکلی که در این روش رو به رو هستیم مشکل collision یا تصادم است یعنی دو کلید متفاوت داشته باشند ولی آدرس های یکسان تولید شود.

$$\text{collision} : k_i \neq k_j \Rightarrow h(k_i) = h(k_j)$$

پس باید رفع تصادم کنیم، برای این کار دو کار راه کلی وجود دارد (زنجیره سازی، آدرس دهی باز):

زنجیره سازی (Chaining)

Hash Table

1 اشاره گر آدرس	تمامی کلید هایی که آدرس 2 تولید میکنند به صورت پیوندی با هم در خانه دوم جدول ذخیره می شوند
2 اشاره گر آدرس	عنصر اول
3 اشاره گر آدرس	عنصر اخر
m-1 اشاره گر آدرس	در حفره های جدول (خانه های آرایه) داده با کلید ذخیره نمی شود فقط بلگه فقط اشاره گر وجود دارد

-۱ هر حفره (خانه آرایه Hash Table)

یک اشاره گر به ابتدای یک لیست پیوندی است و عناصر تصادفی باهم زنجیر می شوند، مثلا اگر ۱۵ کلید آدرس ۲ تولید کند هر ده کلید در خانه دوم باهم زنجیر می شوند.

برای افزون عناصر به لیست فرقی نمی کند که عنصر جدید ابتدای لیست باشد یا انتهای لیست ولی بهتر است ابتدای لیست باشد چرا که مرتبه زمانی (۱) θ دارد این درحالی است که برای درج در انتهای لیست باید لیست را پیمایش کنیم.

مثال: کلید های زیر از سمت چپ وارد شده، تابع درهم ساز به صورت $h_{(k)} = K \bmod 5$ است و $m=5$.

کلید ها: ۱۷، ۴۳، ۴۶، ۵۲، ۵۸، ۷۴، ۷۶، ۹۱، ۹۳

قدم اول

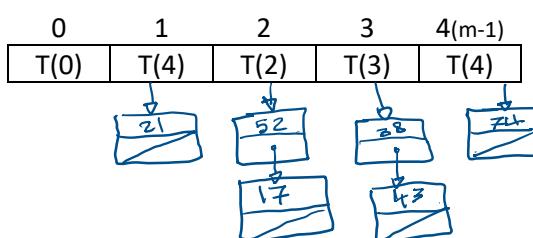
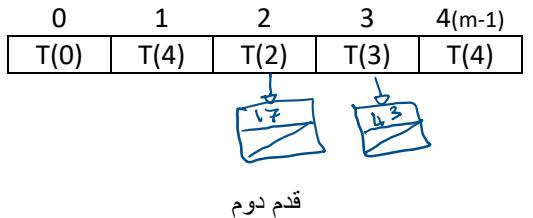
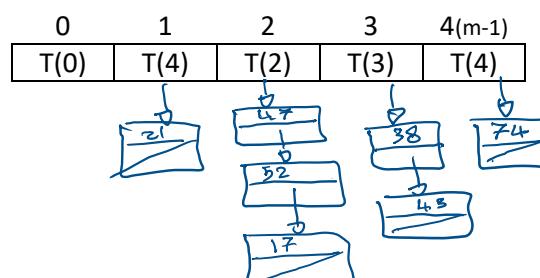
با روش زنجیره سازی کلید های فوق را ذخیره کنید.

اشاره گرها با (i) نشان داده شده اند

1	$17 \bmod 5 = 2$	2	$43 \bmod 5 = 3$
3	$52 \bmod 5 = 2$	4	$21 \bmod 5 = 1$
5	$74 \bmod 5 = 4$	6	$38 \bmod 5 = 3$
7	$47 \bmod 5 = 2$		

عناصر به ترتیب وارد می شوند

قدم آخر



در این روش اگر تعداد عناصر n باشد و تعداد حفره‌ها (خانه‌های آرایه) m باشد بین n و m رابطه این نیست یعنی ممکن است $n > m$ یا $m = n$ یا ... باشد مثلاً می‌توان ۱۰۰ عنصر را ۴ حفره ذخیره کرد در واقع تعداد عناصر محدودیت ندارد.

پیچیدگی زمانی عملیات‌های دیکشنری (درج، حذف، جست و جو):

درج : (1) θ چون اول لیست عنصر جدید را درج می‌کنیم

حذف : (1) θ در صورتی که لیست دو طرفه باشد. (لیست یک طرفه (n))

جست و جو : منظور از جست جو این است که کلید عنصر به تابع جست و جو داده می‌شود این تابع با مرتبه (1) آدرس حفره را پیدا می‌کند و بعد در لیست بررسی می‌کند که آیا عنصری با کلید داده شده وجود دارد یا نه به علاوه یک هایی که می‌بینید به دلیل این است که با (1) θ تابع درهم ساز صدا زده می‌شود.

بهترین حالت جست و جو (کلید، عنصر اول لیست باشد) : $\theta(1 + 1)$

بدترین حالت (عنصر آخر لیست باشد یا اصلاً نباشد) : $\theta(n + 1)$

حالات متوسط : با فرض یکنواختی تابع درهم ساز به طوری که عناصر در هر خانه به صورت مساوی تقسیم

شده است (یعنی در هر حفره $\frac{n}{m}$ فاکتور لود α عنصر هست) پس با این فرایض حالت متوسط $(1 + \alpha)$

قضیه "در روش زنجیره سازی با تابع درهم ساز یکنوا، جست و جو موفق در حالت متوسط از مرتبه $(1 + \alpha)$ است" را اثبات کنید.

تعداد مقایسه بینی جست و جوی حد متفاوت بدایی است با تعداد عناصره قبل $1 + \alpha$
 عنصره قبل از عنصر مورد نظری هستند که بعاز عنصر مورد نظر بین α (عنصری اول جمع می‌شوند)
 احتمال α که α عنصر α را می‌ک لیست قرار بگیرد $\frac{1}{m}$ است، فرض کنند C رفتار دلیل حفره
 هم‌لاید قدرت (مقدار m) $\leq \alpha$ ، m انتخاب خار که $\frac{1}{m}$ احتمال طوفانی قرار بگیرد C هم
 قرار بگیرد است.

$$\begin{aligned}
 & \text{ما } n \text{ عنصر داریم که بدترین بدل } \sum_{i=1}^n \alpha_i \text{ بین } \alpha \text{ و } \alpha + \alpha \text{ بود } \text{ تعداد مقایسه حابهای جست و جوی } \\
 & \text{بینبرایست یک بخلافه تعداد عنصره بیناز } \alpha \text{ بین } \alpha + \alpha \text{ (قبل از } \alpha \text{ قرار گذاشت) عنصر } a_1, a_2, \dots, a_n \text{ با } \\
 & \text{بعاز بیناز } a_i \text{ بین } \alpha \text{ و } \alpha + \alpha \text{ از کدام از عنصرها } a_1, a_2, \dots, a_n \text{ با احتمال } \frac{1}{m} \text{ می‌سته قرار گذاشت } a_i \\
 & \text{قیاره طور (ست زیر مقدار می‌گیرند) سه تعداد مقایسه برای جست و جوی } a_i \text{ بینبرایست } \geq \frac{1}{n} \sum_{i=1}^n (1 + \sum_{j=i+1}^n \frac{1}{m}) \\
 & \text{تعداد مقایسه از } a_i \text{ خود } \underbrace{1 + \sum_{j=i+1}^n \frac{1}{m}}_{\text{تعداد مقایسه از } a_i} \Rightarrow \text{تعداد مقایسه } \underbrace{\frac{1}{n} \sum_{i=1}^n (1 + \sum_{j=i+1}^n \frac{1}{m})}_{\text{تعداد مقایسه از } a_i} \\
 & \text{برای مجموعه کوچک } \underbrace{a_1, a_2, \dots, a_n}_{a_i} \text{ تعداد مقایسه از } a_i
 \end{aligned}$$

$$\begin{aligned} \text{متوسط هفای} &= \frac{1}{n} \sum_{j=1}^n \left(1 + \sum_{i=j+1}^n \frac{1}{m} \right) = \frac{1}{n} \sum_{i=1}^n \left[1 + \frac{1}{m} \sum_{j=1}^n 1 \right] \leq \frac{1}{n} \sum_{j=1}^n \left[1 + \frac{n-i}{m} \right] \\ \text{تفاوت} &= \frac{1}{n} \left[\sum_{i=1}^n 1 + \sum_{i=1}^n \frac{n}{m} - \sum_{i=1}^n \frac{i}{m} \right] \geq \frac{1}{n} \left[n + \frac{n^2}{m} - \frac{1}{m} \left(\frac{n(n+1)}{2} \right) \right] = 1 + \frac{n}{m} - \frac{n+1}{2m} \\ 1 + \frac{n}{m} - \frac{n+1}{2m} &\geq 1 + \frac{1}{2} \frac{n}{m} - \frac{1}{2} \frac{1}{m} + 1 + \frac{\alpha}{2} - \frac{\alpha}{n} = \Theta(1+\alpha) \quad \leftarrow \text{خطه کردیم} \end{aligned}$$

Hash Table

k_i, d_i

عناصر داخل
حفره های ذخیره
شده اند

احتمال بر خورد دو عنصر $\frac{1}{m}$ است.احتمال متوسط بر خورد با فرض تابع درهم سازی یکنوا $\binom{n}{2}$ است.

آدرس دهی باز (Open Addressing)

- عناصر داخل حفره ها ذخیره سازی می شوند، که اگر دو عنصر تصادف کردند (collision) برای عنصر دوم حفره خالی پیدا میکنیم.

- برای پیدا کردن حفره خالی برای عنصر تصادفی سه روش وجود دارد : ۱- کاوش خطی (linear probing) ۲- کاوش مربعی (square probing) ۳- کاوش دوبل (double probing)

۱- کاوش خطی (linear probing) (تست خیر و کنکوری)

از محل برخورد به سمت انتهای آرایه حفره هارا یکی کی بررسی میکند اگر حفره خالی پیدا نشد کاوش را از ابتدای آرایه ادامه می دهد، اگر m کاوش کردیم و حفره خالی پیدا نشد اعلام مکنیم سریز Overflow دنباله کاوش هابه صورت $h'_{(k,i)} = (h(k) + i) \bmod m$ for $i = 0, 1, \dots, m-1$ می باشد.

$$h'_{(k)}, h'_{(k+1)}, h'_{(k+2)}, \dots, m-1, 0, 1, \dots, h'_{(k-1)}$$

مثال : کلید های زیر از سمت چپ وارد شده ، تابع درهم ساز به صورت $h_{(k)} = K \bmod 5$ است و $m=5$.

قدم اول

0	1	2	3	4(m-1)
	17	23		

کاوش خطی برای 43 از عنصر 2 تا انتهای اولین خانه
خالی 43 در آن قرار میگیرد یعنی خانه 4 ام

قدم دوم

1	$17 \bmod 5 = 2$	2	$23 \bmod 5 = 3$
3	$43 \bmod 5 = 2$	4	$33 \bmod 5 = 3$
5	$32 \bmod 5 = 2$		

عناصر به ترتیب وارد می شوند

کلید ها : ۱۷ و ۳۲ و ۳۳ و ۴۲ و ۳۹

0	1	2	3	4(m-1)
33		17	23	43

برای 33 خانه 3 پر است خانه 4 هم پر است در خانه 0 قرار می گیرد

اگر عنصر دیگری نیز وجود داشت اعلام می کردیم overflow برای جست و جو نیست همین رفتار را داریم یعنی مثلا برای ۳۲

اول خانه ۲ را چک میکنیم اگر درست بود که هیچ اگر نه ۳ و ۴ و ۰

و ۱ را بررسی میکنیم تا پیدا شود که در بهترین حالت (۱) θ است

در حالت میانگین هم (۱) θ است ولی در بدترین حالت (۵) θ است.

0	1	2	3	4(m-1)
33	32	17	23	43

برای 32 خانه 2 پر است خانه 4 پر است خانه 0 پر است در خانه 1 قرار می گیرد

key	A	B	C	D	E	F	G
hash	3	5	3	4	5	6	3

مثال : در جدول درهم سازی مقابله کلید و خروجی hash function را داده است این کلید ها به هر ترتیب دلخواه که درج شوند یکی از گزینه های زیر امکان ذخیره اش در آرایه $[T_0 \dots T_6]$ وجود ندارد ، آن را پیدا کنید.(ذخیره سازی با کاوش خطی است)

CGBADEF(۴

BDFACEG(۳

CEBGFDA(۲

EFGACBD(۱

در این سوالات باید دید ابتدا در هر گزینه کدام عناصر در جای خود قرار دارند مثلا در گزینه ۱ A در خانه ۳ قرار دارد بهتر است تحلیل با این عناصر شروع شود.

بررسی گزینه ها :

0	1	2	3	4	5	6
E	F	G	A	C	B	D

(۱) اول A در سه قرار گیرد بعد B وارد شود که در خانه ۵ قرار گیرد بعد C وارد شود که در خانه ۴ قرار گیرد بعد D وارد شود که در خانه ۶ قرار گیرد بعد E وارد شود که در خانه ۰ قرار گیرد بعد F وارد شود که در خانه ۱ قرار گیرد بعد G وارد شود که در خانه ۲ قرار گیرد

0	1	2	3	4	5	6
C	E	B	G	F	D	A

(۲) اول G وارد شود که در خانه ۳ قرار گیرد بعد اگر A وارد شود خانه ۴ می نشیند اگر B وارد شود خانه ۵ می نشیند اگر C وارد شود خانه ۶ می نشیند اگر F وارد شود خانه ۰ می نیشیند که هیچ کدام از این حالت نیست پس گزینه ۲ امکان ندارد اتفاق بیافتد.

(۳) سایر گزینه ها نیز این چنین تحلیلی می شود.

مثال : در مثال قبل کلید ها چند جایگشت برای ورود دارند که گزینه ۱ تولید شود؟

یکی از جایشگت ها خود ABCDEFG است چون A و B هر کدام در جای خود قرار دارند فرقی ندارد کدام اول وارد شوند پس یکی دیگر از جایشگت ها BACDEFG اگر اول A وارد شود در خانه ۳ می نشیند و بعد اگر C را وارد کنیم در خانه مینشیند بعد اگر B را وارد کنیم در خانه ۵ می نیشیند پس جایگشت دیگر ACBDEFG است. جایگشت دیگری وجود ندارد.

مثال : در ۱ مثال قبل کلید ها چند جایگشت برای ورود دارند که گزینه ۳ تولید شود؟

AECGBDF , EACGBDF , ACEGBDF

مثال : در ۲ مثال قبل کلید ها چند جایگشت برای ورود دارند که گزینه ۴ تولید شود؟

اگر گزینه را تحلیلی کنید به ADEFBCG ممکن است برسید که A,D,E,F در جای اصلی خود قرار دارند که به جایشگتی میتوانند بیایند پس !۴! حالت اینجا داریم اما CGB نمیتوانند جایشگت داشته باشند پس همان !۴! حالت داریم

مثال : با توجه به جدول داده شده کلید ها ۷! جایگشت ورودی دارند. چه تعداد جدول درهم سازی مختلف میتوان با این کلید ها ساخت؟ (یعنی با ۷! حالت ورودی به چند شکل متفاوت ذخیره سازی داریم)

برای تحلیل از خانه ای شروع می کنیم که تکلیف مشخص است مثلاً خانه صفر نمیدانیم چه عنصری در آن قرار میگیرد ولی خانه ۳ مشخص است (قانونی ندارد و وابسته به سوال باید به نتیجه برسید ، سوال سختی است)

در خانه ۳ حتماً یکی از A, B, C, D, E, F, G است و ...

0	1	2	3	4	5	6
حالت 3	حالت 2	حالت 1	A, B, C	C, G, D	D, G, B, F	F, E, B, D

$$1 \times 2 \times 3 \times 4 \times 3 \times 3 = 864$$

- ۳- کاوش مربعی :

با فرمول ۱ $h'_{(k,i)} = (h(k) + c_1 i + c_2 i^2) \bmod m$ for $i = 0, 1, \dots, m-1$ دنبال جای خالی میگردد که به ازای c_1, c_2 های متفاوت آدرس خانه های متفاوت به دست می آید به ازای برخی از مقادیر c_1, c_2 آدرس همه حفره ها تولید میشود (بدست آوردن این مقادیر خارج از بحث است) یکی از مقادیر $c_1 = c_2 = \frac{1}{2}$ می باشد که قادر است آدرس تمامی خانه ها را تولید کند.

مثال : بررسی کنید به ازای $m=8$ برای $c_1 = \frac{1}{2}$ $c_2 = \frac{1}{2}$ فرمول $h'_{(k,i)} = (h(k) + \frac{1}{2}i + \frac{1}{2}i^2) \bmod 8$ قادر است آدرس ۱ تا ۷ را تولید کند.

$$\text{برای } m=8 \text{ داریم : } h'_{(k,i)} = \left(h(k) + \frac{1}{2}i + \frac{1}{2}i^2 \right) \bmod 8$$

فرض کنیم یکی از خانه ها مثلاً ۵ پر است و عنصری جدید میخواهد وارد آن شود یعنی $h(k) = 5$ است ، باید دید آیا با فرمول فوق می توان تمامی اندیس های ۰ تا ۷ را تولید کرد .

$$h'_{(k,0)} = h(k) = 5 \quad \text{پر است میرویم حفره بعدی ۵}$$

$$h'_{(k,1)} = \left(5 + \frac{1}{2} + \frac{1}{2} \right) \bmod 8 = 6 \quad \text{مثلاً پر است میرویم حفره بعدی}$$

$$h'_{(k,2)} = (5 + 1 + 2) \bmod 8 = 0 \quad \text{مثلاً پر است میرویم حفره بعدی}$$

$$h'_{(k,3)} = \left(5 + \frac{3}{2} + \frac{9}{2} \right) \bmod 8 = 3 \quad \text{مثلاً پر است میرویم حفره بعدی}$$

$$h'_{(k,4)} = (5 + 2 + 8) \bmod 8 = 7 \quad \text{مثلاً پر است میرویم حفره بعدی}$$

$$h'_{(k,5)} = \left(5 + \frac{5}{2} + \frac{25}{2} \right) \bmod 8 = 4 \quad \text{مثلاً پر است میرویم حفره بعدی}$$

$$h'_{(k,6)} = (5 + 3 + 18) \bmod 8 = 2 \quad \text{مثلاً پر است میرویم حفره بعدی}$$

$$h'_{(k,7)} = \left(5 + \frac{7}{2} + \frac{49}{2} \right) \bmod 8 = 1 \quad \text{همه حفره ها بررسی شد اگر پر باشد سریز میکنیم}$$

در کاوش خطی ما با $clustering$ رو به رو هستیم که این مشکل در کاوش مربعی رفع شده است و به $clustering$ ثانویه تبدیل شده است در روش کاوش دوبل این مشکل نیز بر طرف می‌شود.

۴- کاوش دوبل :

با فرمول $h'_{(k,i)} = (h_1(k) + i \cdot h_2(k)) \bmod m$ for $i = 0, 1, \dots, m - 1$ دنبال جای خالی میگردد چون از دوتابع درهم ساز استفاده میکند $clustering$ اولیه و ثانویه از بین می‌رود.

مثال : فرض کنید $m=13$ و $h_1(k) = k \bmod 13$ و $h_2(k) = 1 + (k \bmod 11)$ Hash Table به صورت زیر می‌باشد.

0	1	2	3	4	5	6	7	8	9	10	11	12
	79			69	98		72		80		50	

کلید ۱۴ در کدام حفره قرار میگیرد (با روش کاوش دوبل) ؟

با توجه به $h_1(k)$ و $h_2(k)$ داریم :

$$h_1(k) = 14 \bmod 13 = 1 \quad 9 \quad h_2(k) = 1 + (14 \bmod 11) = 4$$

برای $h'_{(k,i)}$ داریم :

$$h'_{(k,0)} = (1 + 0 * 4) \bmod 13 = 1 \quad \text{پر است}$$

$$h'_{(k,1)} = (1 + 1 * 4) \bmod 13 = 5 \quad \text{پر است}$$

$$h'_{(k,2)} = (1 + 2 * 4) \bmod 13 = 9 \quad \text{پر است}$$

$$h'_{(k,3)} = (1 + 3 * 4) \bmod 13 = 0 \quad \text{پر نیست پس در این خانه ذخیره می‌شود}$$

نکته : برای آنکه فرمول کاوش دوبل همه حفره هارا بررسی کند ، تابع $h_2(k)$ مقداری باشد که نسبت به m اول باشد. یک راه برای اطمینان از این وضعیت این است که m توانی از ۲ باشد و $(k)h_2$ خروجی فرد تولید کند. یا راه دیگر این است که m اول باشد و $(k)h_2$ خروجی کمتر از m تولید کند (مثل مثال بالا).

مثال : برای جست و جو وقتی کلیدی به تابع درهم ساز داده می‌شود (فرض کنید کلید وجود ندارد) تابع از کجا باید بفهمد کلید وجود ندارد ؟ (کاوش خطی)

وقتی به یک حفره خالی (nil) برسیم به این معنی است که عنصری که جست و جو می‌کنیم در آرایه نیست چون اگر بود در آن خانه خالی ذخیره می‌شد. اگر آرایه به طول کامل پر باشد پس از m کاوش اگر عنصر پیدا نشد یعنی جست و جو ناموفق.

در حذف ممکن است مشکلی پیش بیاید برای این که مشکلی به وجود نیاید عنصری که حذف می‌شود را $deleted$ می‌کنیم نه nil وقتی خانه ای $deleted$ باشد موقع حذف آن خانه پر در نظر گرفته می‌شود ولی موقع درج یا جست و جو آن خانه خالی در نظر گرفته می‌شود.

► قضیه : در درهم سازی زنجیری با فاکتور لود $1 < \frac{n}{m} < \alpha$ و تابع یکنوای ساده متوسط تعداد عمل مقایسه(کاوش) در جست و جوی ناموفق حداکثر $\frac{1}{1-\alpha}$ است.

○ اثبات : فرض کنید می خواهیم کلید k را جست و جو کنیم و قرار است ناموفق باشد . کلید k را می دهیم به تابع درهم ساز این تابع به ما یک آدرس می دهد با احتمال α این آدرس پر است پس تا کنون یک کاوش کرده ایم حال با احتمال α باید بقیم سراغ حفره بعدی که احتمال اینکه حفره اول و دوم پر باشد α^2 است و به همیت ترتیب ... α^3 و (یک دنباله هندسی تشکیل می شود $1 < \alpha$)

$$\alpha + \alpha^2 + \alpha^3 + \dots = \frac{1}{1-\alpha}$$

► قضیه : درج یک عنصر در جدول درهم سازی زنجیری و با تابع یکنوای ساده و $1 < \frac{n}{m} < \alpha$ حداکثر $\frac{1}{1-\alpha}$ مقایسه در حالت متوسط انجام میدهد.

► قضیه : در درهم سازی زنجیری و با تابع یکنوای ساده و $1 < \frac{n}{m} < \alpha$ تعداد کاوش‌های جست و جوی موفق برابر با $\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$ می باشد.

○ اثبات : درج n امین عنصر $\frac{1}{1-\alpha} = \frac{1}{1-\frac{n}{m}} = \frac{1}{\frac{m-n}{m}} = \frac{1}{\frac{n}{m}}$ مقایسه نیاز دارد(آرایه از صفر است) پس درج $i+1$ امین عنصر $\frac{1}{1-\frac{i}{m}}$ مقایسه لازم دارد (جست و جوی موفق مثل درج $i+1$ امین عنصر است) حال برای همه عناصر داریم :

$$\begin{aligned} \sum_{i=0}^{n-1} \frac{1}{1-\frac{i}{m}} &= \sum_{i=1}^n \frac{m}{m-i} = \frac{m}{n} \sum_{i=1}^n \frac{1}{m-i} = \frac{m}{n} \left(\frac{1}{m} + \frac{1}{m-1} + \dots + \frac{1}{m-n+1} \right) \xrightarrow{\text{سری همسازه}} \\ \frac{m}{n} \left(\ln m - \ln(m-n) \right) &= \frac{m}{n} \left(\ln \frac{m}{m-n} \right) \xrightarrow{\alpha=\frac{n}{m}} \frac{1}{\alpha} \ln \frac{1}{1-\alpha} \end{aligned}$$

تابع درهم ساز (Hash function)

پیاده سازی آن بحث ریاضی است که میگذریم

خوب است یکنوا باشد یعنی با احتمال یکسان آدرس حفره هارا تولید کند

خوب است به تمامی ارقام کلید وابسته باشد مثلًا تابع $h(k) = k \bmod m$ یکنواست (هر $h(k)$ گفته شده که یکنواست m اگر به صورت 2^p باشد خوب نیست چون باقی مانده هر عدد به 2^p می شود p بیت سمت راست آن عدد دلیل خوب نبودن این حالت اینکه تابع به تمامی بیت‌های عدد وابسته نیست فقط به m سمت راست وابسته است لذا خوب است m توانی از دو یا نزدیک توانی از دو نباشد.

مثال : فرض کنید کلید ها به صورت رشته ای از کارکتر ها هستند که در مبنای 2^p به عدد طبیعی تبدیل می شود ، در واقع برای هر کارکتر کد اسکی آن را در نظر میگیرم و سپس رشته را به عدد طبیعی تبدیل میکنیم و حاصل می شود کلید آن عنصر ، تابع درهم ساز $m = 2^p - 1$ و $h(k) = k \bmod m$ است ، نشان دهید این تابع در رشته ای که کارکتر های آن جایگشتی از یکدیگر اند را به یک حفره میبرد و تصادم ایجاد می کند.

حل : فرض کنید $p=7$ باشد که $m = 127$ می شود یعنی جدول ما ۱۲۷ خانه دارد همچنین دو رشته داریم که جایگشت هم اند $A=xyz$ و $B=yzx$ طبق سوال A که یک رشته است در مبنای ۱۲۸ در نظر میگیریم و همچنین B :

0	1	...	125	126
		...		

$$A = (xyz)_{128} = x * 128^2 + y * 128^1 + z$$

$$B = (yzx)_{128} = y * 128^2 + z * 128^1 + x$$

حال باید اثبات کنید که اگر اعداد a,b را به تابع $h(k) = k \bmod m$ دهیم در یک خانه قرار میگیرند:

$$h(A) = (x * 128^2 + y * 128^1 + z) \bmod 127 \xrightarrow{\text{بنابر قوانین همنهشتی}} x + y + z$$

$$h(B) = (y * 128^2 + z * 128^1 + x) \bmod 127 \xrightarrow{\text{بنابر قوانین همنهشتی}} y + z + x$$

که مقدار برابری دارند.

درنتیجه این تابع ، زیاد مناسب نیست زیرا کلید های زیادی را در یک حفره می برد.

آنالیز استهلاکی(amortized analysis)

ما n عمل رویه یک ساختمان داده ای انجام میدهیم (مثلاً استک) هر عمل دارای هزینه ای است (هزینه زمانی) می خواهیم میانگین این هزینه های را بدست آوریم و به میانگین بدست آمده هزینه استهلاکی گوییم.

هزینه استهلاکی متوسط هزینه عملیات ها در بدترین حالت است.

محاسبه هزینه استهلاکی روش های متفاوتی دارد. (۱-جمعی(aggregate)، ۲-حسابرسی(accounting))

تجمعی(aggregate)

$$\text{هزینه استهلاکی} = \frac{\text{مجموع کل هزینه ها}}{\text{تعداد عملیات}}$$

مثال: n عمل رویه یک ساختمان داده انجام شده است هزینه عمل i در بدترین حال i^2 است . هزینه استهلاکی عملیات ها را بدست آورید.

هزینه ها به صورت جدول زیر است :

عمل	۱	۲	۳	۴		n
هزینه	۱	۴	۹	۱۶		n^2

$$\text{هزینه استهلاکی} = \frac{1 + 4 + 9 + 16 + \dots + n^2}{n} = \frac{n(n+1)(2n+1)}{6} = \frac{(n+1)(2n+1)}{6} = \theta(n^2)$$

مثال : تعداد 2^n عمل رویه یک ساختمان داده اجرا شده است هزینه عمل i است اگر i توانی از 2 باشد در غیر این صورت هزینه ثابت c است (اگر i توانی از 2 نباشد). مطلوب است هزینه استهلاکی:

هزینه ها به صورت جدول زیر است :

عمل	۱	۲	۳	۴	۵	۶	۷	۸	...	2^n
هزینه	۱	۲	c	۴	c	c	c	۸	...	2^n

عمل انجام داده ایم 2^n

$$\text{هزینه استهلاکی} = \frac{1 + 2 + c + 4 + c + c + c + 8 + \dots + 2^n}{2^n} = \frac{1 + 2 + 4 + 8 + \dots + 2^n}{2^n} + \frac{c + c + c + \dots + c}{2^n}$$

در عبارت (*) تعداد جملات $n+1$ تا n توان های 2 از ۰ تا n ادامه دارند پس $n+1$ جمله داریم ،

$$(*) : \frac{1 + 2 + 4 + 8 + \dots + 2^n}{2^n} = \frac{\frac{1(1 - 2^{n+1})}{1 - 2}}{2^n} =$$

تعداد کل جملات ما 2^n بوده است پس تعداد جملاتی که هزینه انها c است $(n+1) - n = 1$ است. پس داریم :

$$(**) : \frac{c + c + c + \dots + c}{2^n} = \frac{(2^n - (n+1)) * c}{2^n}$$

$$\xrightarrow{(*), (**)} \text{هزینه استهلاکی} = \frac{2^{n+1} - 1}{2^n} + \frac{(2^n - (n+1)) * c}{2^n} = \frac{2^{n+1} - 1 + 2^n * c - (n-1)c}{2^n} \\ = \frac{2^n(c+2) - (n-1)*c - 1}{2^n} = c + 2 - \frac{(n-1)*c - 1}{2^n} = c + 2 = \theta(1)$$

مقدار $\frac{(n-1)*c-1}{2^n}$ بسیار کوچک است

مثال : آرایه ای پویایی داریم به اندازه 1 ، که میخواهیم n بار عمل درج را در آن انجام دهیم ، هزینه هر عمل درج 1 واحد است البته اگر در اثر عمل درج آرایه پر شود یک آرایه به اندازه دو برابر آرایه فعلی می سازیم و عناصر آرایه فعلی را به نیمه اول آرایه جدید منتقل می کنیم و آرایه فعلی را از بین می برمی . هزینه انتقال برای هر عنصر 1 واحد است ، هزینه استهلاکی هر عمل درج را بدست آورید .

درج اول دو واحد هزینه دارد 1 هزینه برای درج ولی آرایه پر میشود یک هزینه هم برای انتقال لازم است یعنی آرایه a_1

به صورت مقابل است $\boxed{a_1} \quad \boxed{}$

درج دوم ۳ واحد هزینه دارد 1 واحد برای درج عنصر دوم ، دو واحد برای انتقال به آرایه جدید چون در

$\boxed{a_1} \quad \boxed{a_2} \quad \boxed{} \quad \boxed{}$

عصر درج عنصر دوم آرایه پر می شود

به همین ترتیب داریم :

$\boxed{a_1} \quad \boxed{a_2} \quad \boxed{a_3} \quad \boxed{}$

درج عنصر سوم 1 واحد هزینه دارد.

$\boxed{a_1} \quad \boxed{a_2} \quad \boxed{a_3} \quad \boxed{a_4} \quad \boxed{} \quad \boxed{} \quad \boxed{}$

درج عنصر چهارم ۵ واحد هزینه دارد.

...

عنصر	۱	۲	۳	۴	۵	۶	۷	۸	۹
هزینه	۲	۳	۱	۵	۱	۱	۱	۹	۱

طبق رابطه بالا هزینه عنصر i ام برابر با $1 + 2^{n-i}$ است اگر ا توانی از دو باشد ، در غیر این صورت هزینه ثابت 1 است ،

در مثال قبل برای 2^n اثبات کردیم در اینجا می توان از 1 صرف نظر کرد چون تاثیر بسیار کمی نسبت به 2^n دارد پس اینجا هم هزینه استهلاکی $C+2$ است یعنی : هزینه استهلاکی برابر است با 3^n .

حسابرسی(accounting)

در این روش فرض میکنیم هزینه استهلاکی X است، سپس ثابت میکنیم که اگر هزینه استهلاکی X باشد تا بی نهایت می توان عملیات هارا انجام داد.

مثال : آرایه i پویایی داریم به اندازه 1 ، که میخواهیم n بار عمل درج را در آن انجام دهیم ، هزینه هر عمل درج 1 واحد است البته اگر در اثر عمل درج آرایه پر شود یک آرایه به اندازه دو برابر آرایه فعلی می سازیم و عناصر آرایه فعلی را به نیمه اول آرایه جدید منتقل می کنیم و آرایه فعلی را از بین می برمی . هزینه انتقال برای هر عنصر 1 واحد است ، هزینه استهلاکی هر عمل درج را بدست آورید .

فرض میکنیم هزینه استهلاکی 3 است باید ثابت کنیم اگر هزینه استهلاکی 3 باشد تا همیشه می توان عمل درج را انجام داد

a_1 درج اول دو واحد هزینه دارد 1 هزینه برای درج ولی آرایه پر میشود یک هزینه هم برای انتقال لازم است یعنی آرایه به صورت مقابل است

$a_1 \quad a_2$ درج دوم 3 واحد هزینه دارد 1 واحد برای درج عنصر دوم ، دو واحد برای انتقال به آرایه i جدید چون در

a_1	a_2		
-------	-------	--	--

عصر درج عنصر دوم آرایه پر می شود

1 واحد ذخیره از قبل داشتیم که با آن عنصر اول را انتقال دادیم ، 2 واحد هم برای درج و انتقال عنصر دوم خرج کردیم پس یک واحد باقی می ماند .

$a_1 \quad a_2 \quad a_3 \quad$ درج عنصر سوم 1 واحد هزینه دارد و 2 واحد ذخیره می شود به همراه یک واحد از قبل پس 3 واحد ذخیره داریم .

$a_1 \quad a_2 \quad a_3 \quad a_4 \quad$ درج عنصر 4 ام 5 واحد هزینه دارد 3 واحد ذخیره داشتیم 3 واحد برای عنصر چهارم گرفتیم که 5 واحد آن خرج می شود و 1 واحد می ماند .

این روند تا آخر ادامه دارد تا توان های دو پول ذخیره میکنیم و در توان های دو پول خرج می کنیم .

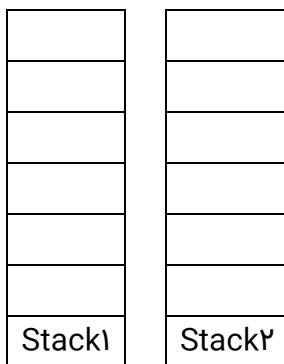
مثال : یک عدد k بیتی با مقدار اولیه صفر داریم . می خواهیم این عدد k بیتی باینری را n مرتبه increment هزینه واقعی هر افزایش برابر است با تعداد بیت هایی که تغییر می کند ، هزینه استهلاکی را بدست آورید . (k بسیار بزرگ است)

به ازای هر افزایش دو تومان نیاز داریم یک تومان برای increment و یک واحد به بیتی که یک است قرض داده می شود که اگر نیاز بود صفر شود ، پس هزینه استهلاکی 2 است . به جدول زیر توجه کنید

عنصر	٠٠٠٠١	٠٠٠١٠	٠٠٠١١	٠٠١٠٠	٠٠١٠١	٠٠١١٠	٠٠١١١	٠١٠٠٠
هزینه	1	2	1	3	1	2	1	4	...
مقدار جدید	2	2	2	2	2	2	2	2	...
ذخیره	1	1	2	2	3	3	4	2	...

تا همشه میتوان با دو واحد هزینه استهلاکی increment کرد .

مثال : با دو تا پشته یک صف بسازید و عملیات add و delete را پیاده سازی کنید و بررسی کنید که اگر n عمل add و delete جمعاً انجام دهیم . بیشترین هزینه یک عمل چقدر است؟ منظور از هزینه یعنی push و pop که هر push و pop یک واحد هزینه دارند ، هزینه استهلاکی هر عمل حداقل چقدر است؟



برای پیاده سازی یک صف با دو استک add و delete را به صورت زیر تعریف میکنیم :

stack1 کردن عنصر به **Add**

اگر $stack2$ خالی بود تمام $stack1$ را در $stack2$ push کن و بعد از $stack2$ خالی نبود فقط pop کن .

روش تجمعی : حداکثر هزینه زمانی است که ما از n عملی که مجازیم انجام دهیم ۱ عمل add کنیم و ۱ عمل delete کنیم (چراکه با $n-1$ عمل عناصر به استک ۱ وارد می شود و باید عمل حذف عناصر از استک ۱ pop و در استک ۲ پوش می شوند) بنابراین این عمل حذف بیشترین هزینه را دارد.

بدترین هزینه : هزینه این حذف

$$(n-1)+(n-1)+1=2n-1 = \text{هزینه این حذف}$$

در رابطه بالا $n-1$ عنصر باید از $stack1$ پاپ شود و $n-1$ عنصر پاپ شده در $stack2$ پوش شود و یک عنصر از استک ۲ خارج شود.

$$(n-1)*(1+(2n-1)) = \text{جمع کل هزینه ها در بدترین حالت}$$

برای جمع هزینه ها در بدترین حالت ما $n-1$ عنصر در $stack1$ پوش کردیم و میخواهیم یک عنصر را حذف کنیم به صورت فوق. قابل ذکر است که اگر به هر صورت دیگر عملیات هارا انجام دهیم کمتر از $3n-2$ خواهد شد ، بنابراین :

$$\frac{\text{جمع کل عملیات ها در بدترین حالت}}{n} = \frac{3n - 2}{n} \cong 3 = \text{هزینه استهلاکی هر عمل}$$

در روش تجمعی ، هزینه بدست آمده برای مجموع درج و حذف است اما با روش حسابرسی میتوان برای درج و حذف به طور جداگانه هزینه استهلاکی حساب کرد.

حسابرسی: طبق سوال هزینه هر push یک تومان هزینه دارد و طبق تعریف ما هر add یک push است پس هر عمل add یک واحد هزینه اصلی دارد اما برای حذف معلوم نیست چقدر هزینه می شود چراکه وابسته به عناصر موجود در خودش و عناصر موجود در $stack1$ است .

اگر فرض کنیم هزینه درج ۳ است ، که یک تومان آن برای push شدن در $stack1$ خرج می شود و ۲ تومان آن ذخیره می شود پس هر عنصر موجود در $stack1$ دو تومان پول دارد.

برای حذف در اثر انتقال عناصر از $stack1$ به $stack2$ آن دو تومانی که ذخیره کرده اند از دست می رود چون یک عمل pop و یک عمل push انجام می شود و ما یک تومان برای حذف کمیود داریم پس هزینه استهلاکی حذف ۱ تومان است. بنابراین هزینه استهلاکی برای درج ۳ و برای حذف ۱ است

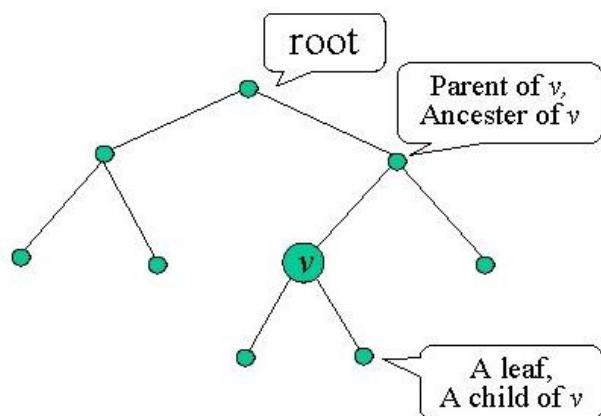
(Tree) درخت

تعریف (گسته ای) : گراف همبند فاقد سیکل (درخت آزاد)

در این درس ما با نوع خاصی از درخت روبه رو هستیم (درخت ریشه دار).

درخت ریشه دار : در ابتدا یک گره داریم به نام ریشه درخت root و بعد به یک سری گره افزایش می شود که بع آن میگوییم زیر درختان ریشه (v) که این زیر درختان خودشان درخت ریشه دار هستند.

در این درس منظور از درخت ، درخت ریشه دارد است.



شکل 1

با توجه به شکل ۲ ، تعاریف زیر را داریم :

۱- والد ، parent :

• A و B و C است (A بی والد است)

• E و D است (D parent B)

• و

۲- همزاد ، sibling :

• sibling E و D هستند.

• sibling F و G هستند.

• هر گره ای که parent یکسان دارد.

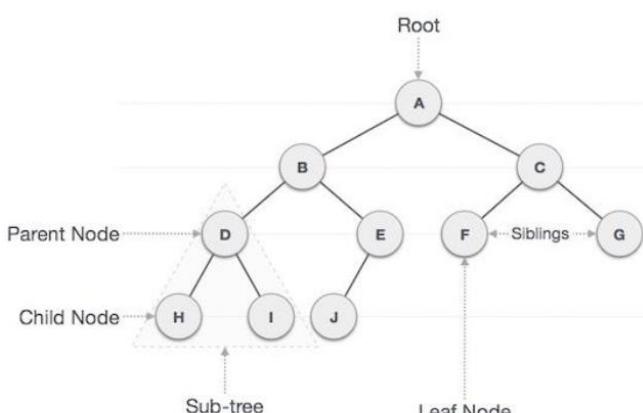
۳- اجداد ، ancestor :

• خود node و بالا هاش

• اجداد H,D,B,A :

• اجداد J,E,B,A :

اجداد مخصوص : بالایی هاش



شکل 2

✓ اجداد محسن L : E,B,A

۴- نوادگان ، Descendent :

• خودش و پایین هاش

• نوادگان B : B,D,E,H,I,J,L

❖ نوادگان محسن : پایین هاش

✓ نوادگان B : D,E,H,I,J,L

۵- درجه ، Degree :

• تعداد فرزندان (child)

• درجه ۲ B است

• در درخت شکل ۲ بیشنه درجه های ۳ است پس درخت را "درخت سه تابی ای درجه ۳" گویند.

توجه کنید که اگر درخت آزاد بود درجه تعداد یال های هر node بود مثلًاً برای B درجه ۳ می شد.

۶- درخت k تابی (k-ary)

• درختی که یک node آن حداکثر k فرزند دارد. ($\text{max degree} = k$)

۷- برگ ، leaf ، گره خارجی ، external node :

• گره (node) هایی که درجه آن ها صفر است.

• L,I ، H,I ، leaf یا گره خارجی هستند.

۸- گره داخلی ، internal node :

• نود هایی که درجه آنها صفر نیست

• تمامی گره ها باجز L,I,H,I گره داخلی است.

اگر درخت تک node داشته باشد ، ریشه گره خارجی می شود.

۹- ارتفاع ، Height :

• تعریف کتاب clrs (تعریف معیار) : ارتفاع هر node فاصله آن نود تا دورترین برگ موجود در زیر درختان آن node است.

❖ ارتفاع برگ ها صفر است

❖ ارتفاع D,C,E یک است

ارتفاع B ع دو است

ارتفاع A (ارتفاع درخت) : ۳ است

- تعريف ارتفاع در کتاب Horwick (تعریف قبلی معیار است): ارتفاع درخت تعداد سطوح درخت است

ارتفاع درخت شکل ۲ ، ۴ است چون ۴ سطح دارد

: depth ، ۱۰ عمق ،

- عمق هر نود فاصله نود تا ریشه (تقرباً عکس ارتفاع) (سطح (level) یکی بیشتر از عمق است)

عمقش ۰ است

عمق ۱ دارند B,C

عمق دو D,E,F,G

عمق درخت (H,I,J) ۳ دارند.

نکته: فقط عمق درخت با ارتفاع آن برابر است.

مثال: درست عبارت "در هر درخت حداقل ۴ نودی، حداقل یک نود وجود دارد که عمش با ارتفاعش برابر است" را بررسی کنید.



اسباب ایس : شلن نقفن

مثال: یک درخت k تایی که x سطح دارد:

الف) حداقل چند node دارد?

ب) حداقل چند برگ دارد?

ج) حداقل چند گره داخلی دارد?

الف) حداقل زمانی رخ می دهد که درخت پر باشد.

$$1 + K + K^2 + K^3 + \dots + K^{x-1}$$

(سطح می بینید از ارتفاع است) سطح x ام

فرزندهای سطح k زیر است

بنابر این برای درخت x سطحی حداقل تعداد گره ها برابر است با:

$$\max(nodes) = 1 + k + k^2 + k^3 + \dots + k^{x-1} = \frac{1 - k^x}{1 - k}$$

ب) حداکثر تعداد برگ ها برابر است با حداکثر نود های سطح آخر درخت پر یعنی k^{x-1}

$$\text{ج) تعداد کل برگ ها منتهی برگ ها یعنی: } 1 + k + k^2 + k^3 + \dots + k^{x-2} = \frac{1-k^{x-1}}{1-k}$$

اگر تعداد برگ ها برابر با l باشد برای درخت k تایی تعداد نود های داخلی برابر است با $\frac{1-l}{1-k}$

توصیه می شود فرمول های گفته شده را حفظ نکنید و درک کنید.

مثال: یک درخت k تایی به ارتفاع 7 دارای حداکثر 279936 برگ است، این درخت حداکثر چند گره داخلی دارد؟

ارتفاع 7 یعنی 8 سطح داریم. پس $k^7 = 279936$ برگ (گره خارجی داریم) داریم

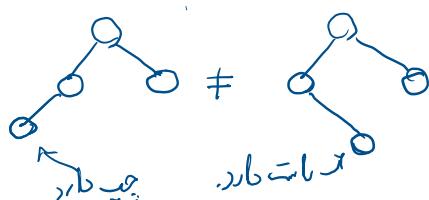
$$k^7 = 279936 \Rightarrow k = ?$$

محاسبه k بدون ماشین حساب دشوار میرسد ولی توجه کنید، k یک عدد زوج است و کمتر از ۱۰ چون k^7 شش رقمی است، همچنین از ۸ هم کمتر است چون $2^{21} = 65536$ باید $8^7 = 2^{21}$ که اگر این عدد را به توان ۵ برسانیم خیلی بزرگ تر از عدد 279936 می شود همچنین دو رقم سمت راست 279936، ۳۶ است پس به سادگی میتوان تشخیص داد $k = 6$ می باشد.

$$\text{گره خارجی: } \frac{1-279936}{-5} = \frac{279935}{5} = 55987 \text{ که برابر است با 55987}$$

از مهمترین درخت های ریشه دارد درخت دودویی است.

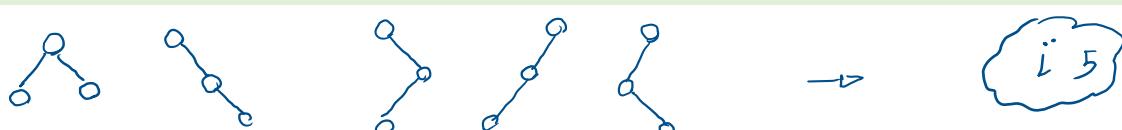
درخت دودویی (Binary Tree)



درختی که هر نodus حداکثر دو فرزند دارد (یا ندارد یا یکی دارد یا دو تا)

در این درخت فرزند چپ و راست متفاوت اند، دو درخت رو به رو متفاوت اند.

مثال: با ۳ نود چند درخت دودویی متفاوت وجود دارد؟ (از لحاظ توپولوژی)

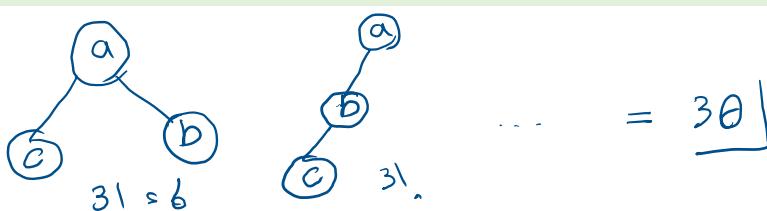


تعداد درخت دودویی با n گره برابر است با عدد C_n کاتالان (اثبات در گستته):

C_n = \frac{1}{n+1} \binom{2n}{n}

خیلی از شمارش ها به عدد C_n کاتالان منجر می شود، مثل پوش و پاپ های استک و

مثال: با 3 نود چند درخت دودویی برچسب دار وجود دارد؟ (برچسب دار یعنی برای هر node اسم بگذارید)

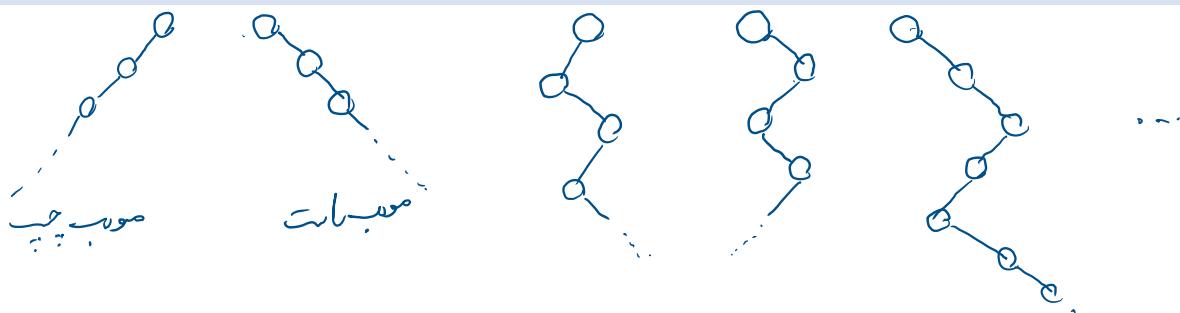


تعداد درخت دودویی برچسب دار با n گره برابر است با C_n برابر عدد C_n کاتالان (اثبات در گستته):

n \cdot C_n = \frac{n}{n+1} \binom{2n}{n}

مثال : با n نود چند درخت دودویی با بیشترین ارتفاع وجود دارد ؟

فقط درخت های مورب چپ و راست نیستند که بیشترین ارتفاع را می سازند علاوه بر آنها حالت های دیگری نیز هست :

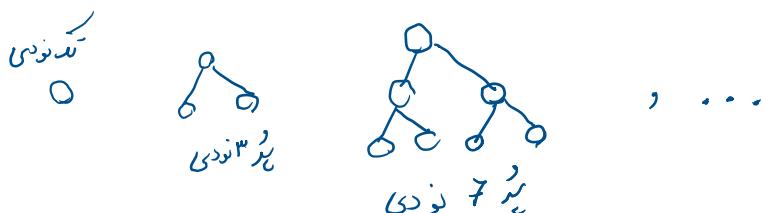


بیشترین ارتفاع زمانی رخ میدهد که در هر سطح فقط یک node وجود داشته باشد ، گره اول ۱ حالت دارد ولی بقیه گره ها

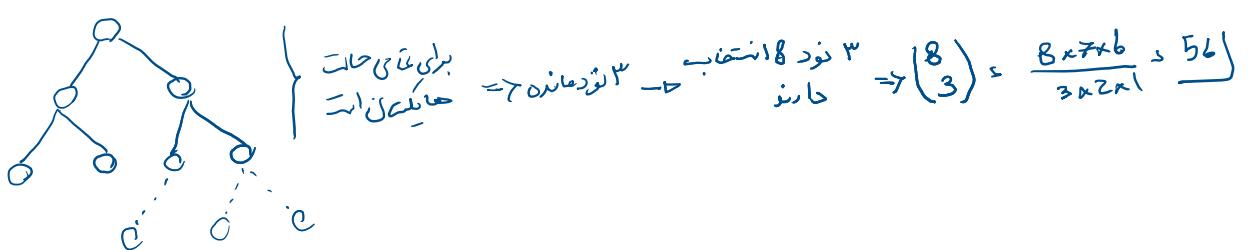
$$2 \text{ حالت دارد. یعنی } 1 * 2 * 2 * \dots * 2 = 2^{n-1}$$

درخت دودویی پُر (Full Tree)

درخت که تک فرزندی ندارد و برگ ها هم سطح اند ، درخت پر با x سطح $1 - 2^x$ نود دارد.



مثال : با ۱۰ تا نود چند تا درخت دودویی وجود دارد که به جز سطح آخر سایر سطوح پر باشند؟



مثال : با ۱۰۰۰ نود چند تا درخت دودویی وجود دارد که همه سطوح بجز آخرین سطوح پر باشد.

$$\begin{aligned} & \text{اعداد سری: } 1 + 2 + 4 + 8 + \dots + 2^k \leq 1000 \Rightarrow 2^k - 1 < 1000 \Rightarrow 2^k < 1001 \Rightarrow k \leq 9 \\ & \text{نفریه: } 2^k = 512 \Rightarrow \text{مجموع نودها} = 511 + 489 = 1000 \Rightarrow \text{نفریه: } 489 \\ & \text{سطح آخر ۹ تا بکار رفته: } 512 - 489 = 23 \end{aligned}$$

مثال : با n نود درخت دودویی حداقل چند سطح (level) دارد؟

حداکثر مشخص است اگر هر سطح یک node داشته باشد آنگاه حداقل n سطح داریم
حداقل زمانی رخ میدهد که تمامی سطح‌ها بجز سطح آخر پر باشد، مانند سوال قبل داریم :

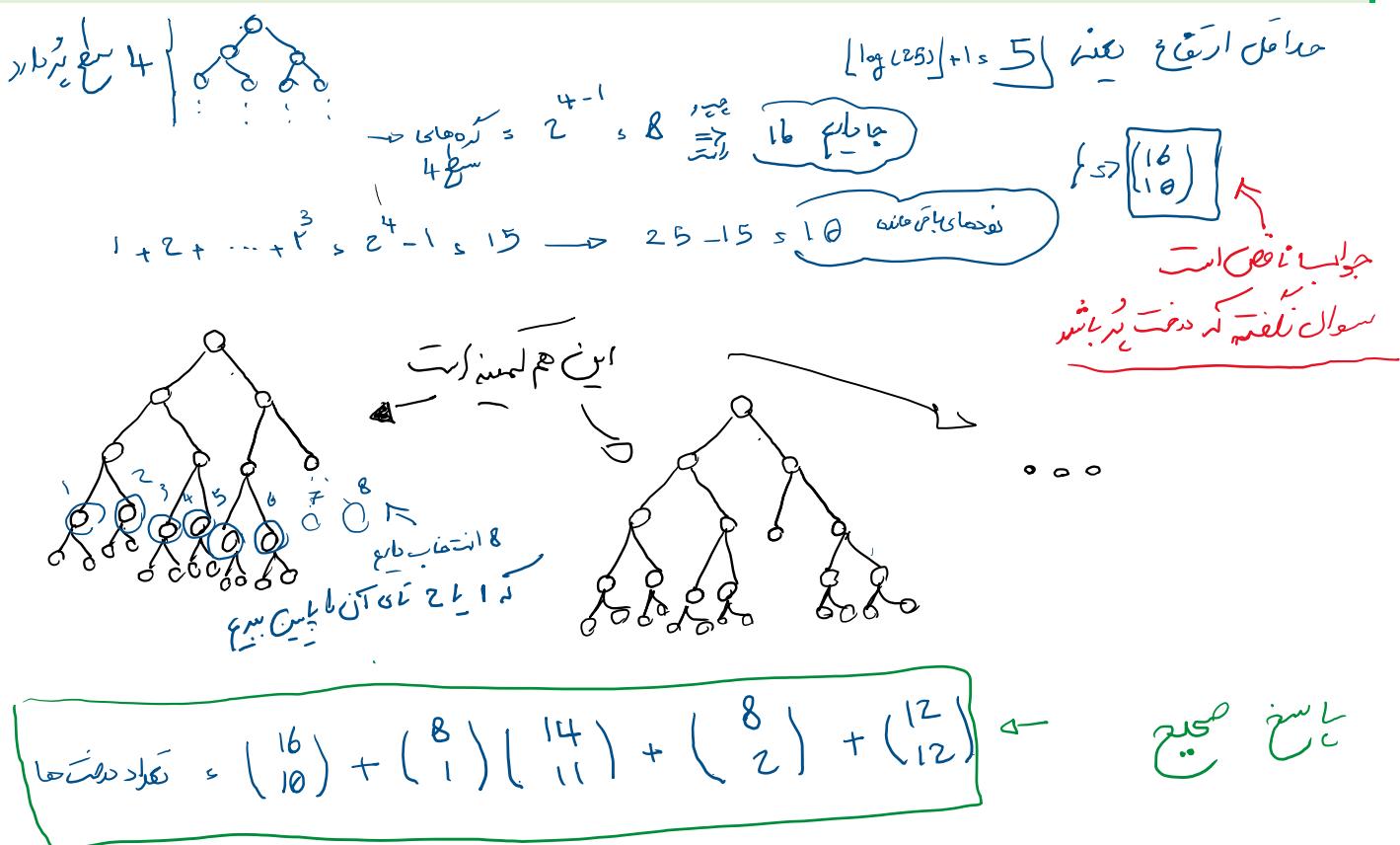
$$1 + 2 + 2^2 + 2^3 + \dots + 2^{x-1} \leq n \rightarrow 2^x - 1 \leq n \rightarrow 2^x \leq n + 1 \rightarrow x \leq \log(n + 1)$$

توجه کنید که x تمامی سطوح بجز سطح آخر را دربرمی‌گیرد (به سوال قبل رجوع کنید) پس پاسخ باید با ۱ جمع شود.

$$x = \lfloor \log n \rfloor + 1 = \lceil \log(n + 1) \rceil$$

حداقل ارتفاع درخت دودویی با $\lceil \log n \rceil + 1$ گره است و حداقل n است.

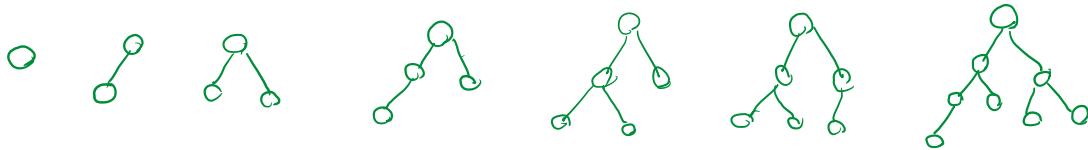
مثال : با ۲۵ گره چند درخت دودویی با ارتفاع کمینه می‌توان ساخت ؟



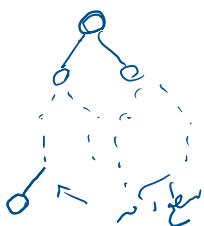
درخت دودویی کامل

تعریف ۱ : درخت دودویی فاقد تک فرزندی

تعریف ۲ (تعریف مورد استفاده) : درختی که تا سطح یکی مانده به آخر پُر است و در سطح آخر گره ها از سمت چپ چیده می شوند.



مثال : درخت دودویی کامل که x سطح دارد ، حداقل و حداقل‌تر چند node دارد ؟



حداقل زمانی رخ می دهد که درخت تا سطح $1-x$ ام پر باشد و در سطح آخر ۱ نود داشته باشد داریم :

می دانیم تعداد برگ ها برای درخت پر k^{x-1} است که برای درخت دودویی $k=2$ و برای سطح $1-x$ ام

تعداد برگ ها برابر است با k^{x-2} مجموع تعداد برگ های تا سطح $1-x$ ام بعلاوه یک تعداد حداقل

نود هاست

$$\min(\text{nodes}) = (1 + 2 + 4 + \dots + 2^{x-2}) + 1 = 2^{x-1}$$

حداکثر زمانی رخ می دهد که درخت کاملاً پر باشد.

$$\max(\text{nodes}) = (1 + 2 + 4 + \dots + 2^{x-1}) = 2^x - 1$$

مثال : چند درخت کامل دودویی پر با x سطح وجود دارد ؟

درخت کامل است ، یعنی تا سطح $1-x$ ام پر است که سطح $1-x-1$ ام ۲ نود دارد برای سطح آخر هر نودی که اضافه کنیم یک درخت جدید ایجاد می شود ، که می توان 2^{x-1} نود اضافه کرد پس 2^{x-1} درخت کامل میتوان ایجاد کرد.

مثال : درخت کامل دودویی با 1000 نود چند سطح دارد ؟

$$1 + 2 + 4 + \dots + 2^{x-2} = 2^{x-1} - 1 \leq 1000 \rightarrow x = \log[1001] + 1 = 10$$

یا درجه اولین توان دو بعد از عدد نود ها ، یا همان ارتفاع . minimum

مثال : درخت کامل دودویی با 1000 نود چند برگ دارد ؟

تا سطح $1-x$ ام پر است داریم :

$$1 + 2 + 4 + \dots + 2^{x-2} = 2^{x-1} - 1 \leq 1000 \rightarrow x = \log[1001] + 1 = 10$$

پس ۱۰ سطح داریم : سطح یکی مانده به آخر یعنی سطح ۹ ام $= 256$ برگ دارید از طرفی مجموع برگ ها تا سطح ۹ ام برابر است با $511 = 511 - 511 = 489 = 2^9 - 1$ تا سطح ۸ ام $= 2^8 = 256$ برگ دارید از طرفی مجموع برگ ها تا سطح ۸ ام $= 489$ برگ دارید از سمت چپ دو به دو در درخت قرار میگیرند یعنی تا نود $= \left\lceil \frac{489}{2} \right\rceil = 245$ برگ در سطح ۸ ام داریم آنرا با $245 = 111 + 111 = 222$ برگ داریم (نصف کل گره ها)

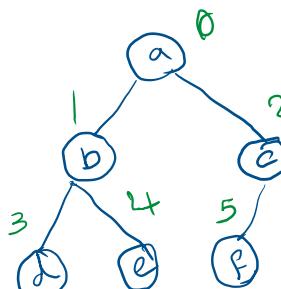
تعداد برگ های درخت کامل با n نود برابر است با $\left\lceil \frac{n}{2} \right\rceil$ و تعداد گره داخلی برابر است با $\left\lfloor \frac{n}{2} \right\rfloor$

مثال : چند درخت کامل دودویی با 1000 نود وجود دارد ؟

بنابر مثال های قبلی ، ۱۰ سطح دارد ، که سطح آخر ۴۸۹ نود دارد و این ۴۸۹ نود فقط به یک شکل می توانند بنشینند پس یکی.

نمایش و پیاده سازی درخت دودویی با آرایه

اگر درخت کامل باشد از ریشه از چپ به راست گره ها را شماره گذاری میکنیم و هر شماره را در اندیس های آرایه ذخیره میکنیم



عنصر ۰ بی پدر است و برای بدست آوردن پدر سایر عناصر کافی است یکی از ایندکس آن در آرایه کم کنیم و جزء صحیح نصفش را بدست آوریم ، برای بدست آورد فرزند چپ کافی است ۲ برابر یک ایندکس را با یک جمع کنیم و برای فرزند راست دو برابر یک اندیس را با ۲ جمع کنیم.

۰	۱	۲	۳	۴	۵
a	b	c	d	e	f

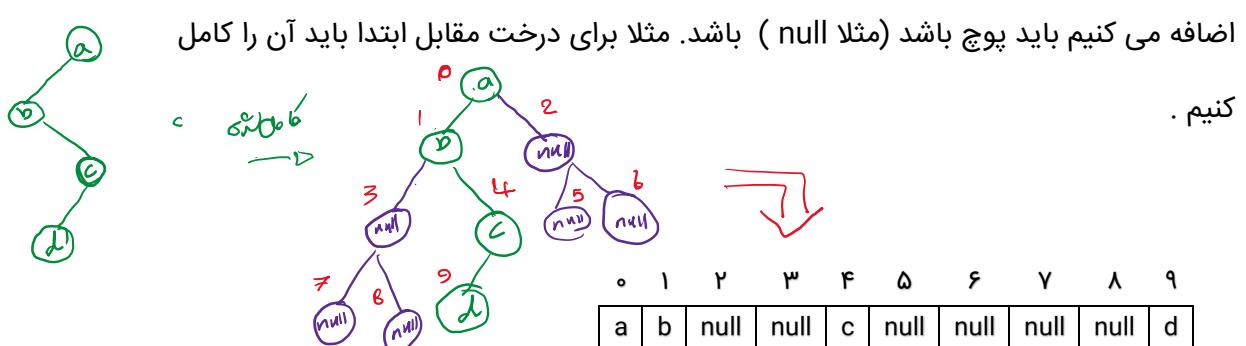
طبق آرایه بالا داریم :

$$\theta_{(1)} \text{ با مرتبه} \left\{ \begin{array}{ll} parent(i) = \lfloor \frac{i-1}{2} \rfloor & i \neq 0 \\ LeftChild(i) = 2.i + 1 & 2.i + 1 \leq n \\ RightChild(i) = 2.i + 2 & 2.i + 2 \leq n \end{array} \right.$$

اگر درخت کامل نباشد ، ابتدا باید درخت را کامل کنیم ، برای کامل کرد از بالا شروع میکنیم و کامل میکنیم ، گره هایی که

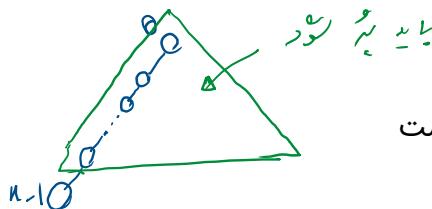
اضافه می کنیم باید پوچ باشد (متلا null) باشد. مثلا برای درخت مقابله ابتدا باید آن را کامل

کنیم .



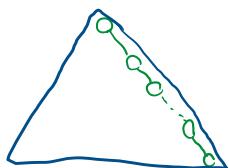
توجه کنید روش آرایه برای درخت کامل خوب است ولی برای درخت هایی که کامل نیست زیاد مناسب نیست.

مثال : درخت مورب چپ نورد ۱۱ نودی با آرایه ، آرایه اش حداقل چه اندازه ای دارد.



باید $n-1$ سطح از این درخت را پر کمیم تا کامل شود برای پر کردن $n-1$ سطح به $1 - 2^{n-1}$ نود نیاز است و یک نود هم در آخر داریم پس آرایه ای با طول 2^{n-1} نیاز است

مثال : درخت مورب راست نورد n نودی با آرایه ، آرایه اش حداقل چه اندازه ای دارد.



باید درخت تماماً کامل شود ، n نود داریم که راست نورد است یعنی هر سطح یه نود دارد پس n سطح داریم که مجموع نود ها برابر است با $1 - 2^n$ نود نیاز است.

شکل کلاس :

```
#include <iostream>
#include<string>
#include<cstring>
using namespace std;
class Tree
{
private:
    int TreeMaxNode;
    char* tree;
public:
    Tree(int TreeMaxNode);
    ~Tree();
    void AddRoot(char key);
    void SetLeftChild(char key, int parent);
    void SetRightChild(char key, int parent);
    string ToString();
};
```

سازند و مخرب کلاس :

```
Tree::Tree(int TreeMaxNode) :TreeMaxNode(TreeMaxNode)
{
    tree = new char[TreeMaxNode];
    memset(tree, '\0', TreeMaxNode);
}
Tree::~Tree()
{
    delete tree;
}
```

افزودن ریشه :

```
void Tree::AddRoot(char key)
{
    if (tree[0] != '\0') throw string("Tree already had root");
    else tree[0] = key;
}
```

افزودن نود سمت چپ :

```
void Tree::SetLeftChild(char key, int parent)
{
    if (tree[parent] == '\0') throw string(("Can't set child at") + to_string((parent * 2) + 1) + (" , no parent found"));
    else tree[(parent * 2) + 1] = key;
}
```

افزودن نود سمت راست :

```
void Tree::SetRightChild(char key, int parent)
```

```
{
    if (tree[parent] == '\0') throw string(("Can't set child at" + to_string((parent * 2) + 2) + " , no parent found")
);
    else
        tree[(parent * 2) + 2] = key;
}
```

تبدیل درخت به استرینگ :

```
string Tree::ToString()
{
    string res = "";
    for (int i = 0; i < TreeMaxNode; i++)
    {
        if (tree[i] != '\0')
            res += tree[i];
        else
            res += "null";
    }
    return res;
}
```

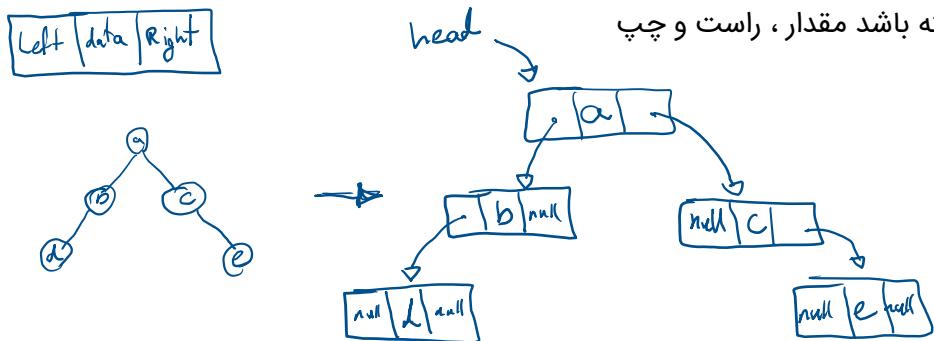
تست کد :

```
int main()
{
    Tree tree (10);
    tree.AddRoot('a');
    tree.SetLeftChild('b', 0);
    tree.SetRightChild('c', 1);
    tree.SetLeftChild('d', 4);
    cout << tree.ToString();
    return 0;
}
```

خروجی کد :

a b null null c null null null null d

نمایش و پیاده سازی درخت دودویی با لیست پیوندی



مثال : در روش لیست پیوندی با n نود تعداد لینک تهی چند تاست.

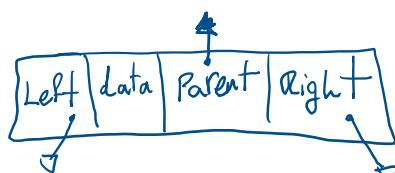
اگر n نود داشته باشیم هر node دو لینک دارد ، چه تهی و چه غیر تهی پس $2n$ لینک داریم ، از طرفی حداقل $n-1$ لینک پر است (در درخت تعداد گره ها از تعداد گره ها یکی کمتر است) یعنی :

$$\text{تعداد لینک تهی} = 2n - (n - 1) = n + 1$$

یافتن آدرس parent نود x در روش پیوندی از مرتبه (n) است که n تعداد گره های درخت است.

یافتن آدرس child های یک نود از مرتبه (1) است.

برای آنکه یافتن parent یک node از مرتبه (1) شود به هر نود در لینک لیست باید یک اشاره گر اضافه شود.



پیمایش درخت

▶ یعنی تمامی گره ها را با نظمی ملاقات کنیم :

◦ دونوع پیمایش داریم :

-۱- پیمایش سطحی (level order)

-۲- پیمایش عمقی (depth order)

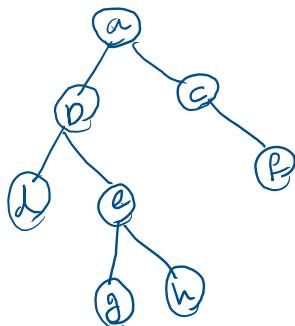
▶ پیمایش سطحی (level order)

- از چپ به راست گره هارا ملاقات میکنیم (به همون ترتیبی که در آرایه ذخیره می کنیم)
- برای پیاده سازی به یک صف نیاز است

- ابتدا گره ریشه را داخل صف قرار میدهیم

- سپس شروع میکنیم هر گره ای که در صف بود را میخوانیم

- به هر گره رسیدیم فرزندانش را داخل صف وارد میکنیم



صف	a	b	c	d	e	...
خروجی	a	b				...

مرتبه این پیمایش $\theta(n)$ است.

◦ پیمایش عمقی

• ۶ نوع پیمایش عمقی داریم که همه آنها به پیشته نیاز دارند.

▪ اگر حرکت سمت راست R و حرکت سمت چپ L و ملاقات با ریشه را با D نشان دهیم ۶ مورد عبارت است از

-۱ - (Preorder) ، DLR (یعنی اول ریشه را ملاقات کن و بعد زیر درخت چپ را با همین روش و بعد راست)

-۲ - (in order)LDR (یعنی اول زیر درخت چپ را پیمایش کن بعد ریشه را ملاقات کن و بعد راست به همین شیوه)

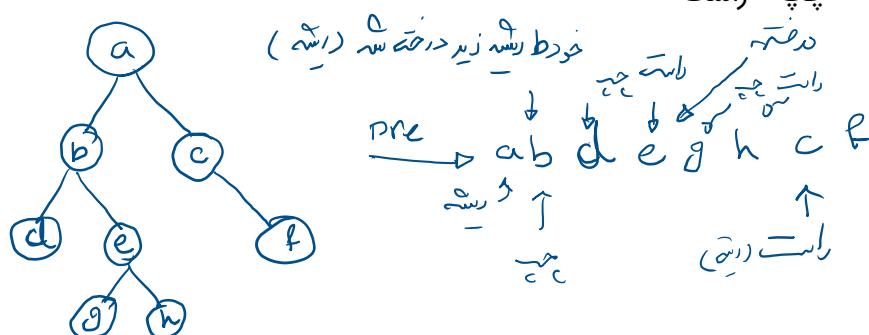
-۳ - (Post order)LRD (یعنی اول زیر درخت چپ را پیمایش کن بعد زیر درخت راست به همین شیوه را پیمایش کن بعد ریشه را ملاقات)

-۴ - (Preorder reverse)RLD (معکوس ۱)

-۵ - (in order reverse)RDL (معکوس ۲)

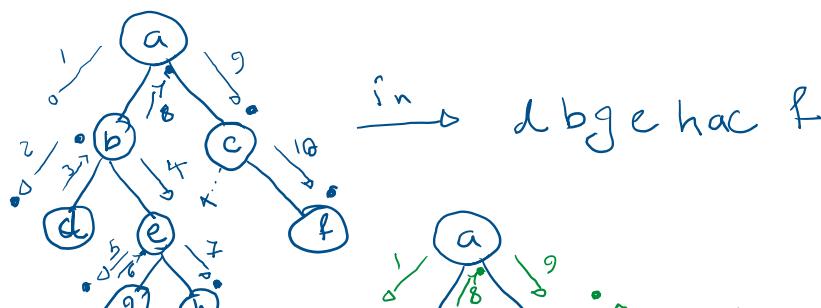
-۶ - (Post order reverse)DRL (معکوس ۳)

- ۱- پیمایش DLR ، Rیشه - چپ - راست :Preorder



۲- پیمایش In order ، LDR ، چپ-ریشه - راست

به هر درختی می رسمیم میرویم چپ

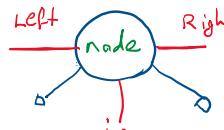


۳- پیمایش Post order ، LRD ، چپ- راست - ریشه

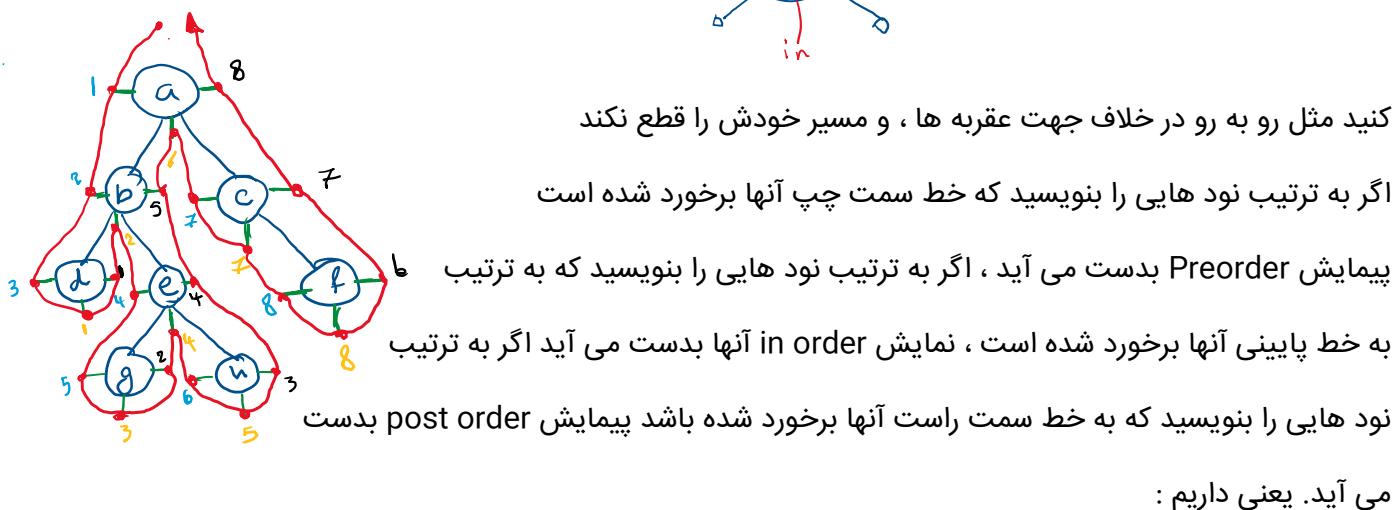


روش ساده برای بدست آوردن پیمایش ها :

حال در درخت از ریشه شروع کنید و در لایه نود ها حرکت



هر نود را به صورت مقابل قطعه بندی کنید



می آید. یعنی داریم :

$$\text{+ preorder} = abdegfhc$$

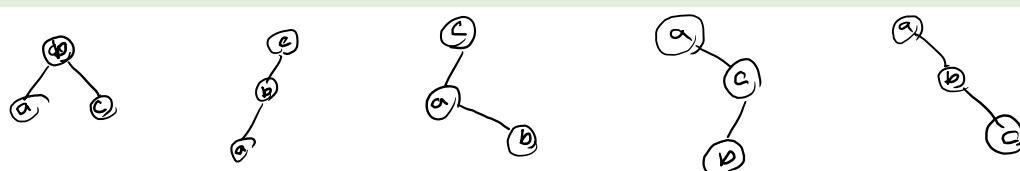
$$\text{+ inoerder} = dbgrhacf$$

$$\text{+ postorder} = dghebfca$$

► در سه پیمایش preorder , in order , post order ترتیب ملاقات برگ ها یکسان و از چپ به راست است.

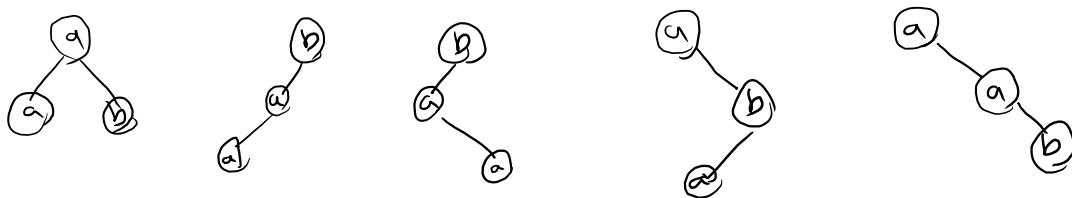
► با داشتن یک پیمایش و بدون اطلاعات اضافی ، درخت منحصر به فرد نیست.

مثال : اگر پیمایش ما in order به صورت abc باشد ، درخت مربوطه را رسم کنید.



با داشتن یک پیمایش حاوی n برچسب و بدون اطلاعات اضافی به تعداد کاتالان تا $\frac{1}{n+1} \binom{2n}{n} = C_n$ درخت می توان کشید

مثال : اگر پیمایش ما in order به صورت aab باشد ، درخت مربوطه را رسم کنید.



► با داشتن پیمایش های in,level یا post,in یا in,in منحصر به فرد است ، یعنی اگر دو پیمایش داشته باشیم که یکی از آنها in order باشد درخت منحصر به فرد است.

مثال : با توجه به پیمایش های زیر ، درختی رسم کنید

In: gcbedfajih

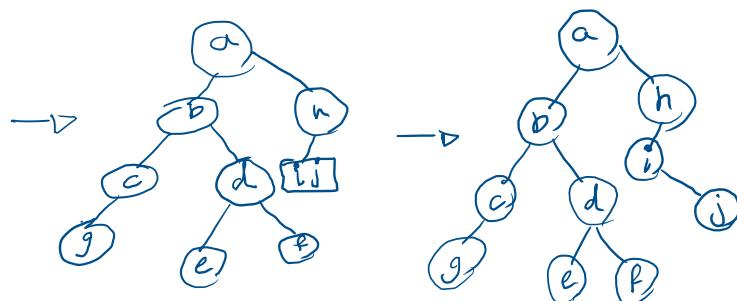
Post : gcefdbjih

در post order یا LRD آخرین گره ریشه است یعنی a پس ریشه را پیدا کردیم حال با توجه به in order متوجه می شویم نود های gcbedf در چپ a قرار دارند و ijh در راست a قرار دارند

همین عملیات را به صورت بازگشتی انجام می دهیم یعنی در بین نود های gcbedf نودی

که در post order راست تر است ریشه بقیه است یعنی b و با توجه به In order نود های gc سمت چپ b و نود های edf سمت راست b قرار دارند به همین ترتیب داریم

با این ریشه مشخص می شود با in order فرزندان چپ و راست



مثال : با توجه به پیمایش های زیر درخت رسم کنید.

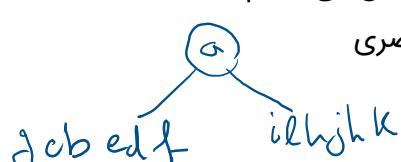
In : gcbedfailjhk

Level : abhdikgefjl

در level order اولین عنصر ریشه است و با توجه به in order سمت چپ و راست گره را تشخیص می دهیم که سمت

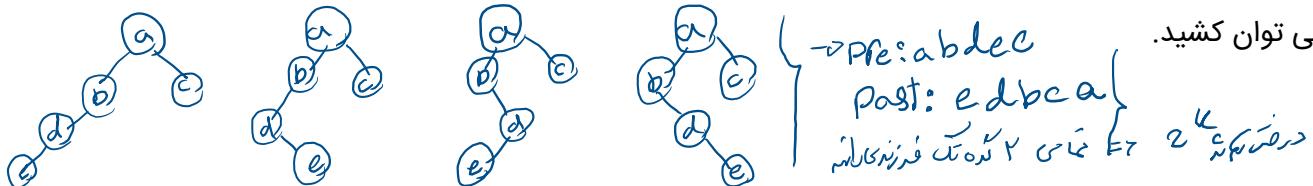
راست a نود های gcbedf و سمت چپ a نود های ijhjk قرار دارند. از میان gcbedf اولین عنصری

که در level order (از سمت چپ) دیده میشود b است یعنی b ریشه بقیه است که چپ



و راست b از inorder $a b c d e f g h i j k l$ تشخیص داده می شود مشخصاً $g c$ سمت چپ و $d e f$ سمت راست آن قرار دارند. برای $i n j g k a$ نیز همین طور است اولین عنصری که در level order دیده می شود h است که h ریشه بقیه است و راست و چپ آن از $i n$ مشخص می شود و به همین ترتیب.

➤ با داشتن پیمایش های pre , post , $a b c \dots k$ تا گره تک فرزندی وجود داشته باشد آنگاه k^2 تا درخت دودویی متفاوت می توان کشید.



➤ با داشتن پیمایش های pre , post , $a b c \dots k$ اگر گره تک فرزندی وجود نداشته باشد درخت منحصر به فرد است.

➤ با داشتن پیمایش pre یا post (یکی شون) و مشخص بودن برگ ها اگر تک فرزندی نداشته باشیم ، درخت منحصر به فرد است.

➤ تنها نوع درختی که پیمایش های pre و in یکسان دارند درخت مورب راست است.

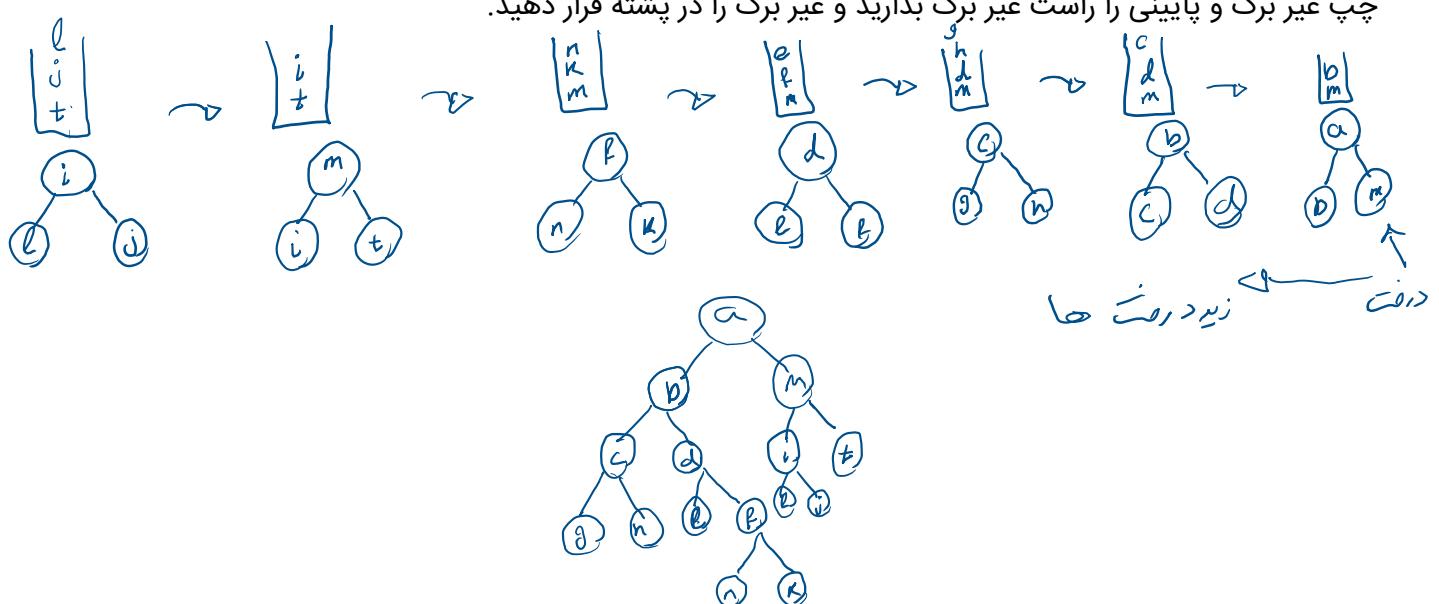
➤ تنها نوع درختی که پیمایش های post و in یکسان دارند درخت مورب چپ است.

➤ در درخت هر درخت دودویی تعداد برگ(n_0) از تعداد گره دو فرزندی(n_2) یک واحد بیشتر است. $n_0 = n_2 + 1$

مثال) با توجه به پیمایش pre داده شده و مشخص بودن برگ ها می دانیم تک فرزندی نداریم ، درخت را رسم کنید.) عناصری که زیر آنها خط کشیده شده است برگ هستند)

Pre : a b c g h d e f n k m i l j t

یک پشته در نظر بگیرد از راست برگ ها را در پشته push کنید ، اگر به غیر برگ رسیدید ، دو تا از پشته بردارید و بالایی را چپ غیر برگ و پایینی را راست غیر برگ بذارید و غیر برگ را در پشته قرار دهید.



مثال : با توجه به پیمایش ها :

Pre : a b c g h d e f k m i j l t

Post : h g c e k f d b l j i t m a

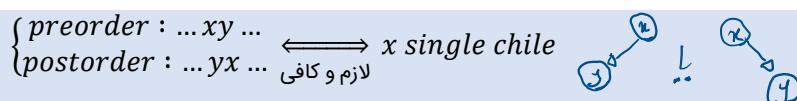
الف) چه تعداد درخت میتوان کشید؟

ب) چه تعداد برگ و گره دو فرزندی وجود دارد؟

ج) برگ ها و گره های دو فرزندی را مشخص کنید

د) یکی از درخت های ممکن را بکشید .

الف) برای پاسخ به این بخش باید بدانیم چند گره تک فرزند داریم که تعداد درخت های می شود تک فرزند² برای تخصیص این کار داریم :



گره ای که در preorder ابتداء می آید parent و گره دوم child است.

طبق نکته فوق c و f و a و z تک فرزندی است یعنی $32 = 2^5$ درخت می توان کشید

Pre : a b c **g** h d e **f** k m **i** j l t

Post : h **g** c e **k** f d b **l** **j** i t m a

ب) طبق الف ما ۵ تک فرزندی داریم که طبق فرمول گره های تک فرزندی c و g و f و a و z هستند که ۵ عدد می باشند و $n_0 = n_2 + 1$ نتیجه میگیریم ۴ دو فرزندی داریم ، ۴ تا تک فرزندی ، ۴ تا دو فرزندی بقیه یعنی ۵ تا برگ است.

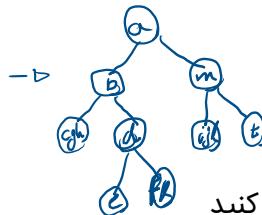
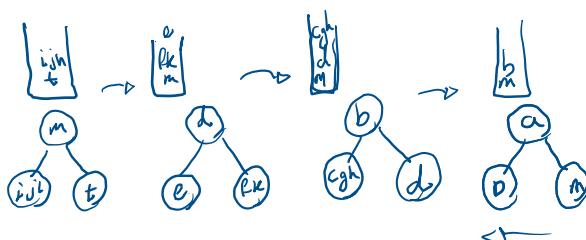
بین برگ ها و تک فرزندی های ارتباطی وجود ندارد و لزوماً برابر نیستند.

ج) در نمایش t a l t pre : a b c g h d e f k m i j l t راست ترین برگ درخت است یعنی بعد از t دیگر برگی وجود ندارد که میتوان از post : h g c e k f d b l j i t m a نتیجه گرفت که برگ m,a که post : h g c e k f d b l j i t m a آخرین عنصر ریشه است a و اولین عنصر چپ ترین برگ درخت نیستند همچنین در pre : a b c g h d e f k m i j l t می توان نتیجه گرفت که بدان معنی است قبل از آن برگ نیست با توجه به t یعنی h که برگ نیستند از طرفی c و g و f و a و z ها تک فرزندی هستند پس بین h e k d b l t برگ ۵ باید پیدا کنیم که دو برگ آن پیدا شده است ، می دانیم که برگ ها در pre و post به یک ترتیب می آیند کافی است از بین این عناصر عناصری که در pre و post به یک ترتیب اند را پیدا کنیم یعنی t h e k l برگ های ما هستند.

د) در این سوال که تک فرزندی دارم ، ابتدا تک فرزندی ها را ادغام میکنیم یعنی و زیر برگ ها خط می کشیم :

Pre : a b (cgh) d e (fk) m (ij) l t

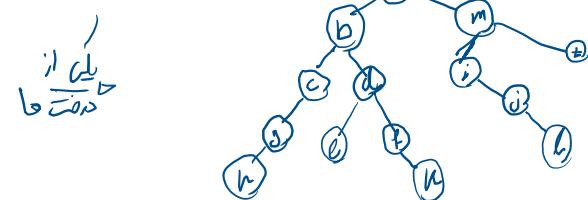
حال ما تک فرزندی ندارم و pre داریم و مانند مثال قبل رسم می‌کنیم:



از بین انهایی که ادغام شده است

به هر شکلی که دوست یکی را رسم کنید، توجه کنید

۳۲ درخت می‌توان رسم کرد.



پیاده‌سازی (کد) درخت با لیست پیوندی و پیمایش‌ها

ساختار کلاس و نود:

```
#include <iostream>
#include<string>
#include <stack>
using namespace std;
struct TreeNode
{
    int parentKey;
    TreeNode* leftChild = nullptr;
    TreeNode* rightChild = nullptr;
};
class Tree
{
private:
    TreeNode* root = nullptr;
    void ToStringInorder(TreeNode* node, string& result);
    void ToStringPostorder(TreeNode* node, string& result);
    void ToStringPreorder(TreeNode* node, string& result);
public:
    Tree();
    ~Tree();
    void Addroot(int parentKey);
    void SetLeftChild(int key, int parent);
    void SetRightChild(int key, int parent);
    //string ToStringLevelorder();
    string ToStringInorder();
    string ToStringPostorder();
    string ToStringPreorder();

    TreeNode* SearchNode(int key);
};
```

سازنده و مخرب کلاس:

```
Tree::Tree()
{
}
Tree::~Tree()
{
    delete root;
}
```

افزودن ریشه:

```
void Tree::Addroot(int parentKey)
{
    root = new TreeNode();
    root->parentKey = parentKey;
}
```

: post order پیمایش

```
void Tree::ToStringPostorder(TreeNode* node, string& result) {
    if (node != nullptr)
    {
        ToStringPostorder(node->leftChild, result); // left
        ToStringPostorder(node->rightChild, result); // Right
        result += to_string(node->parentKey); // Root
    }
}
string Tree::ToStringPostorder()
{
    string res = "";
    ToStringPostorder(root, res);
    return res;
}
```

پیمایش inorder:

```
void Tree::ToStringInorder(TreeNode* node, string& result)
{
    if (node != nullptr)
    {
        ToStringInorder(node->leftChild, result); // left
        result += to_string(node->parentKey); // Root
        ToStringInorder(node->rightChild, result); // Right
    }
}
string Tree::ToStringInorder()
{
    string res = "";
    ToStringInorder(root, res);
    return res;
}
```

: pre order پیمایش

```
void Tree::ToStringPreorder(TreeNode* node, string& result)
{
    if (node != nullptr)
    {
        result += to_string(node->parentKey); // Root
        ToStringPreorder(node->leftChild, result); // left
        ToStringPreorder(node->rightChild, result); // Right
    }
}
string Tree::ToStringPreorder()
{
    string res = "";
    ToStringPreorder(root, res);
    return res;
}
```

جست و جو نود : (به صورت inorder غیر بازگشتی برای جست و جو پیمایش میکنیم)

```
TreeNode* Tree::SearchNode(int key)
{
    // use Inorder Traversals for search
    TreeNode* tmp = root;
    stack<TreeNode*> stack;
    while (true)
    {
        while (tmp != nullptr)
        {
            stack.push(tmp);
            tmp = tmp->leftChild;
        }
        if (!stack.empty())
        {
```

```

        tmp = stack.top();
        stack.pop();
        if (tmp->parentKey == key)
        {
            return tmp;
        }
        tmp = tmp->rightChild;
    }
    else return nullptr;
}
}
}

```

افزودن فرزند چپ :

```

void Tree::SetLeftChild(int key, int parent)
{
    TreeNode* tmp = SearchNode(parent);
    if (tmp != nullptr)
    {
        if (tmp->leftChild == nullptr)
        {
            TreeNode* newnode = new TreeNode();
            newnode->parentKey = key;
            tmp->leftChild = newnode;
        }
        else throw exception();
    }
    else throw string("parent not found");
}

```

افزودن فرزند راست :

```

void Tree::SetRightChild(int key, int parent)
{
    TreeNode* tmp = SearchNode(parent);
    if (tmp != nullptr)
    {
        if (tmp->rightChild == nullptr)
        {
            TreeNode* newnode = new TreeNode();
            newnode->parentKey = key;
            tmp->rightChild = newnode;
        }
        else throw exception();
    }
    else throw string("parent not found");
}

```

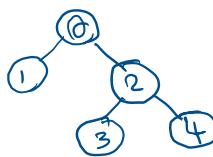
تست کلاس :

```

// Driver Code
int main()
{
    try
    {
        Tree tree;
        tree.Addroot(0);
        tree.SetLeftChild(1, 0);
        tree.SetRightChild(2, 0);
        tree.SetLeftChild(3, 2);
        tree.SetRightChild(4, 2);
        cout << "in Order : " << tree.ToStringInorder() << endl;
        cout << "Post Order : " << tree.ToStringPostorder() << endl;
        cout << "Pre Order : " << tree.ToStringPreorder() << endl;
    }
    catch (...)
    {
        cerr << "Error";
    }
    return 0;
}

```

in Order : 10324
 Post Order : 13420
 Pre Order : 01234



خروجی (حاصل درخت زیر است) :

$$a + b \Rightarrow \begin{array}{c} + \\ \diagup \quad \diagdown \\ a \quad b \end{array}$$

$$\text{not } a \Rightarrow \begin{array}{c} \text{not} \\ \diagup \\ a \end{array}$$

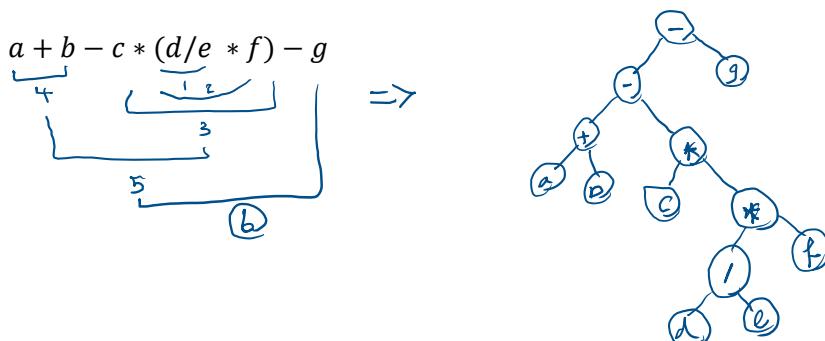
$$a ++$$

$$\begin{array}{c} ++ \\ \diagup \\ a \end{array}$$

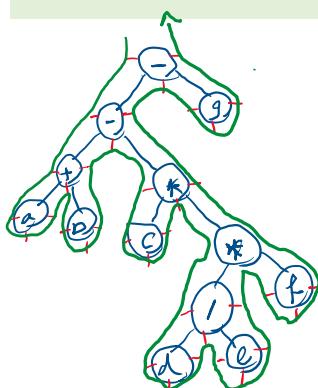
عملگر ها گره های داخلی و عملوند ها برگ هستند

مثال : درخت عبارت $g = a + b - c * (d/e * f) - g$ را رسم کنید.

برای رسم درخت باید الویت بندی کنیم از کمترین الویت (ریشه درخت) شروع میکنیم به اول مینویسیم یعنی از ۶ به ۱ :



مثال : برای درخت رسم شده در مثال قبل پیمایش post order و pre order را رسم کنید



$$\rightarrow \text{pre order} \Rightarrow \text{prefix} = - - + ab * c * / de fg$$

$$\rightarrow \text{post order} \Rightarrow \text{postfix} = ab + c de / fg * * - g -$$

$$\rightarrow \text{in order} \Rightarrow \text{infix} = ((a+b)-(c*(d/e)*f))-g$$

مُثُرَّهٌ
بِهَا نَزَّلْنَا

اگر درخت یک عبارت ریاضی را رسم کنیم ، پیمایش pre order و post order به ما infix می دهد
 دهید ، پیمایش in order مشروط بر پرانتز گذاری هر بخش به ما infix می دهد

مثال : درخت یک عبارت ریاضی دودویی است و دارای 50 نод است ، کدام گزینه درست است ؟

الف) این عبارت عملگر تک عملوندی ندارد

ب) تعداد عملگر های دو عملوندی این عبارت زوج است.

ج) تعداد عملگر های دو عملوندی این عبارت فرد است.

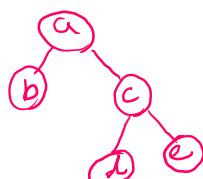
د) این عبارت حداقل یک عملگر تک عملوندی دارد.

اگر گره تک فرزندی نداشته باشیم تعداد گره ها زوج باشد یعنی حداقل یک نود تک فرزندی دارد.

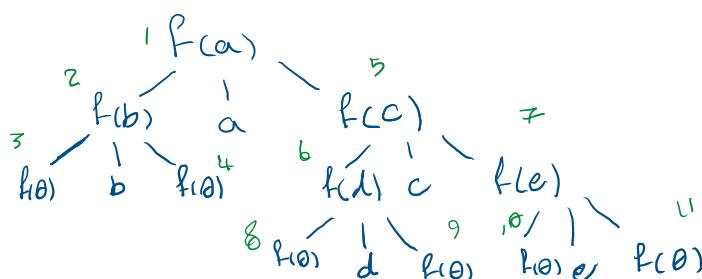
بنابراین نکته بالا با حداقل یک گره تک فرزندی (عملگر تک عملوندی) داریم یعنی گزی

مثال : با توجه به کد مقابل ، اگر این کد را روی درخت زیر اجرا کنیم ، تعداد فراخوانی چند تاست ؟

```
F(x)
{
  If(x != nil)
  {
    F(Left(x))
    Write(data(x))
    F(Right(x))
  }
}
```



با توجه به کد (کد پیمایش in order) چپ - راست را صدا میکند ، یعنی $f(a)$ اول b را صدا میکند بعد a را چاپ میکند بعد c را صدا می زند.



۱۱ فراخوانی انجام شده است ، (تعداد فراخوانی ها برابر است با $call = node + nilnode$ همچنین خروجی به صورت preorder بدهست می آید یعنی خروجی کد بالا به صورت : badce است.

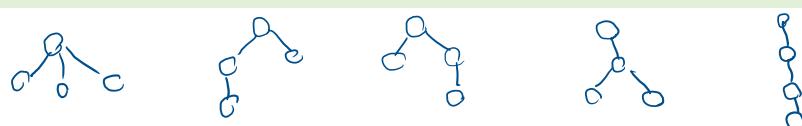
درخت عمومی (General Tree)

درختی ریشه دارد در حالت کلی ، یعنی کی نود به عنوان ریشه دارد و سایر نود ها به تعداد زیر درخت افزایش می شوند که زیر درختان خود اینگونه هستند.



در درخت general فرزند چپ و راست فرقی ندارند :

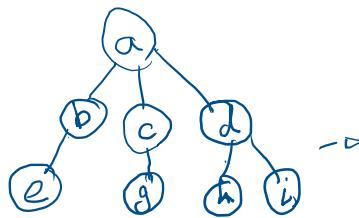
مثال : با 4 نود چند درخت عمومی وجود دارد ؟



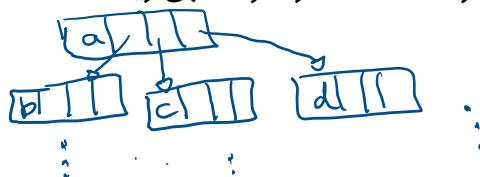
با n نود تعداد درخت های عمومی برابر است با $C_{n-1} = \frac{1}{n} \binom{2(n-1)}{n-1}$ با $n+1$ نود عدد $n!$ م کاتالان می شود.

نحوه نمایش یا پیاده سازی درخت عمومی

- روش اول: اگر هر نود حداقل k فرزند داشته باشد، آنگاه **ساختاری** تعریف میکنیم حاوی k اشاره گر (لینک) باشد (تعمیم دودویی) که هر یک اشاره گر به یک فرزند اشاره کند.

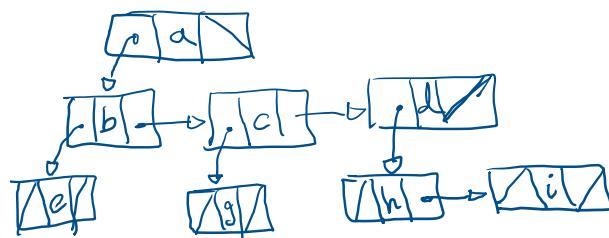
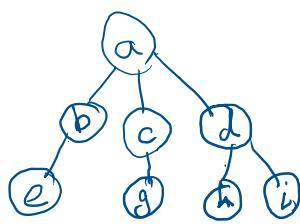
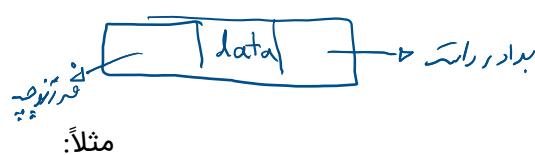


- روش مناسبی نیست چرا که تعداد اشاره گرهای زیاد است.



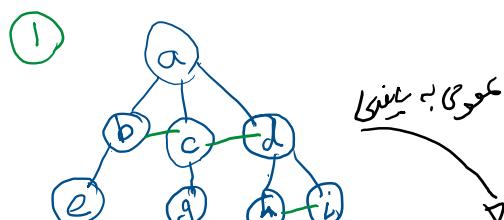
مثالاً

در روش فوق به ازای n نود k تایی $nk - (n - 1)$ لینک تهی خواهد داشت.



در روش فوق به ازای n نود $1 + n$ لینک تهی خواهد داشت.

هر درخت عمومی به روش فوق قابل تبدیل به درخت دودویی است.



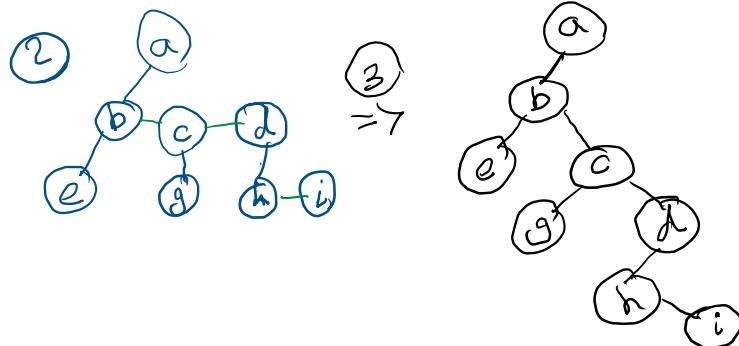
تبدیل درخت عمومی به باینری

الگوریتم زیر را دارم تبدیل درخت عمومی به دودویی:

۱- برادرها را به هم وصل کن

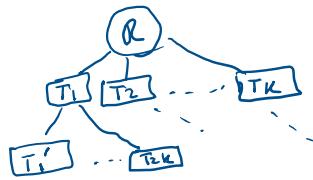
۲- ارتباط فرزند ها بجز چپ ترین را با پدر قطع کن

۳- جهت قشنگ شدن فرزند هارو مرتب کن.



پیمایش درخت عمومی

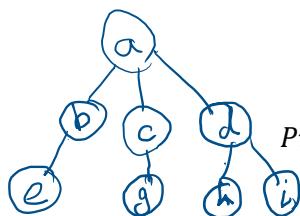
پیمایش درخت عمومی مانند پیمایش درخت دودویی است ، شامل :



هر کدام از $T_1 T_2 T_3 T_4 \dots T_k$ نیر به همین روش $Pre\ Order : RT_1 T_2 T_3 T_4 \dots T_k$ -۱

هر کدام از $T_1 T_2 T_3 T_4 \dots T_k$ نیر به همین روش $Post\ Order : T_1 T_2 T_3 T_4 \dots T_k R$ -۲

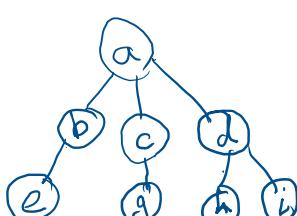
هر کدام از $T_1 T_2 T_3 T_4 \dots T_k$ نیر به همین روش $In\ Order : T_1 R T_2 T_3 T_4 \dots T_k$ -۳



$Pre\ Order : abecgdhi, Post\ order: ebgchida, in\ order = ebagchdi$

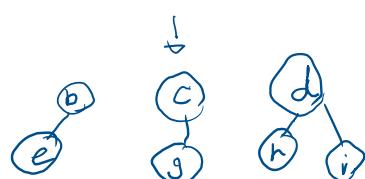
مثالاً برای درخت رو به رو داریم

اگر درخت دودویی معادل درخت عمومی را به صورت pre پیمایش کنیم انگار درخت عمومی را به صورت pre پیمایش کردیم اگر درخت دودویی معادل درخت عمومی را به صورت in پیمایش کنیم انگار درخت عمومی را به صورت post پیمایش کردیم



یک یا تعدادی درخت عمومی ، درخت مقابل جنگلی با یک درخت است

اگر ریشه این درخت یعنی a را حذف کنیم جنگلی با ۳ درخت به وجود می آید

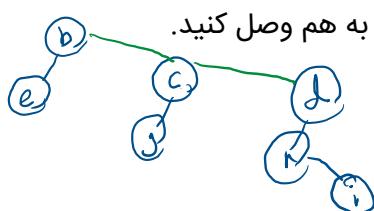


پیمایش جنگل

فقط پیمایش های pre و post دارد به این صورت که به ترتیب درخت ها را از سمت چپ به راست پیمایش عادی کنید و کنار هم بنویسید. برای جنگل فوق $preorder = becgdhi . post\ order = ebgchid$.

تبديل جنگل به درخت دودویی

برای هر درخت مانند درخت عمومی رفتار میکنیم سپس دریش ها را از چپ به راست به هم وصل کنید.



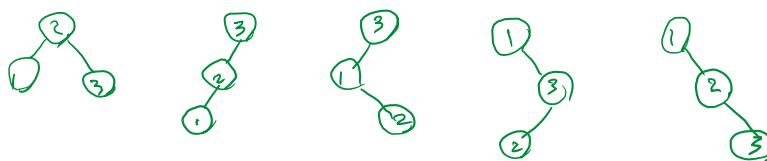
اگر درخت دودویی معادل جنگل را به صورت pre پیمایش کنیم انگار جنگل را به صورت pre پیمایش کردیم اگر درخت دودویی معادل جنگل را به صورت in پیمایش کنیم انگار جنگل را به صورت post پیمایش کردیم (اگر پیمایش جنگل را بخواهد آن را دودویی کنید.)

درخت جست و جوی دودویی (BST)

درخت دودویی، که میتواند تهی باشد، اگر تهی نباشد هر node شامل یک کلید است که کلید ها میتوانند تکراری باشد، کلید هر نод از کلید های زیر درخت چیپش بزرگتر یا مساوی است، و کلید هر نود از زیردرخت راستش کوچکتر یا مساوی است.

برای اینکه بفهمیم درخت دودویی BST است کافی است پیمایش inorder را بدست آوریم اگر صعودی بود درخت bst است این کار مرتبه زمانی $O(n)$ است.

مثال : با سه کلید متمایز چند درخت bst وجود دارد؟



با داشتن n به تعداد کاتالان تا $C_n = \frac{1}{n+1} \binom{2n}{n}$ درخت BST می توان کشید.

تمرین : فرض کنید $C(n)$ تعداد BST با اعداد 1 تا n است، برای C_n فرمول بازگشتی بیابید.

با n نود برای درخت bst حداقل $\log n + 1$ سطح و حداقل 1 سطح دارد.

عملیات های پایه BST

جست و جو ، درج ، \min و \max و succ ، pred و حذف (عملیات دیکشنری)

- جست و جو : جست و جو کلید x در bst، ابتدا x را با ریشه مقایسه میکنیم، اگر مساوی بود که هیچ اگر از کلید ریشه کوچکتر بود در زیر درخت چپ به صورت بازگشتی همین کار را انجام می دهیم و اگر بزرگتر بود در زیر درخت راست.

مرتبه زمانی وابسته به ارتفاع درخت است یعنی از مرتبه $O(h)$ است .

بدترین حالت $\theta(n)$

بهترین حالت $\theta(1)$

حالت متوسط $\theta(\log n)$

- درج : برای درج x در bst ابتدا باید جست و جو کنیم که محل درج را پیدا کنیم.(بینیم از چی بزرگتر و کوچیکتره تا به جای مناسب برسیم)

در حالت کلی ترتیب درج مهم است و خاصیت جا به جایی ندارد، و ممکن است تغییر ترتیب درج شکل درخت را تغییر دهد

اگر عناصر مساوی را بخواهیم درج کنیم فرقی نمی کند چپ درج شود یا راست، اما سعی شود که راهی انتخاب شود تا ارتفاع زیاد نشود، (بهتر است برای هر نود یک کانتر گذاشته شود که مساوی ها در کانتر شمرده شود)

۳- یافتن \min و \max : چپ ترین برگ \min است، و راست ترین برگ \max است.

۴- یافتن predecessor و successor :

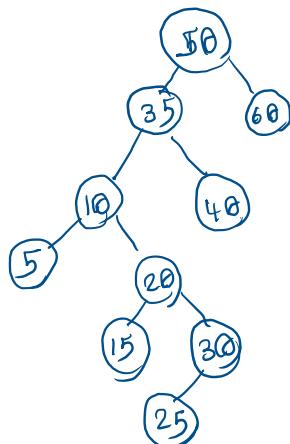
$\text{successor}(x) = x_j$ کوچکترین عدد بزرگتر یا مساوی از x

$\text{predecessor}(x) = x_i$ بزرگترین عدد کوچکتر یا مساوی از x

اگر درخت رو به رو صعودی بنویسیم: $in\ order = 5\ 10\ 15\ 20\ 25\ 30\ 35\ 40\ 50\ 60$

مثالاً: $\text{predecessor}(5) = 10$ عدد predecessor (10) = 15 است و $\text{successor}(10) = 15$ عدد successor (10) = 15 ندارد.

بنابراین برای پیدا کردن predecessor و successor یک راه پیمایش میان ترتیب است که $O(n)$ است که خوب نیست.



اگر x زیر درخت راست داشت، \min زیر درخت راست می شود $\text{succ}(x)$ ولی اگر نداشت اولین جد x که x در زیر درخت چپ آن است می شود $\text{succ}(x)$ که مرتبه این عملیات $O(h)$ منظور از h ارتفاع است.

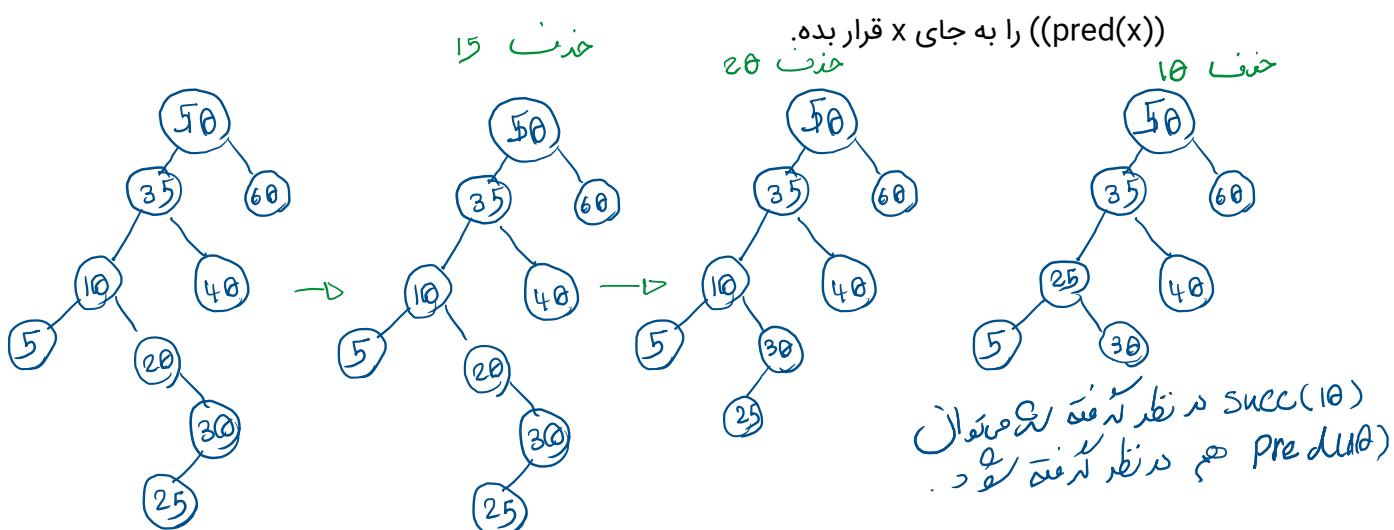
اگر x زیر درخت چپ داشت، \max زیر درخت چپ می شود $\text{pred}(x)$ ولی اگر نداشت اولین جد x که x در زیر درخت راست آن است می شود $\text{pred}(x)$ که مرتبه این عملیات $O(h)$ منظور از h ارتفاع است.

۵- حذف: برای حذف x سه حالت وجود دارد

۱- اگر x برگ بود به راحتی حذف می شود

۲- اگر x یک فرزند داشت (فرزنده ممکن است خانواده داشته باشد)، فرزند را به جای x قرار دهید.

۳- اگر x دو فرزند داشت یا مینیمم زیر درخت راست x ($\text{succ}(x)$) یا ماکزیمم زیر درخت چپ x



در حالت کلی حذف جا به جا پذیر نیست و ترتیب متفاوت حذف ممکن است درخت های متفاوتی ایجاد کند.

ساخت BST

- n کلید داریم و میخواهیم BST بسازیم

- روش اول : اگر همه n کلید یکجا داده شده باشد کافی است ابتدا کلید ها را به صورت صعودی مرتب کنیم سپس کلید وسط(میانه) را ریشه درخت قرار دهیم و حال $\frac{n}{2}$ کلید به زیر درخت چپ و ما بقی به زیر درخت راست وارد می شود که حال با چپ و راست به صورت بازگشتنی این کار را انجام می دهیم.

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + 1 + O(n \log n) = O(n \log n)$$

- روش ۲ ، اگر کلید ها یکی وارد می شوند باید با ورود هر کلید عمل درج انجام دهیم ، یعنی n بار عمل درج را تکرار کنیم.

مثال : کلید های زیر از سمت چپ وارد شده اند ، BST بسازید و تعداد کل مقایسه ها را حساب کنید ، فرض کنید در هر node تعداد تکرار محاسبه می شود. ۳۰, ۱۵, ۵۰, ۷۰, ۲۵, ۱۵, ۳۵, ۱۵, ۵۰, ۲۰

اولین ورودی را ریشه فرض کنید و شروع به عملیات insert کنید اگر تعداد مقایسه ها را بشمارید می شود ۱۷

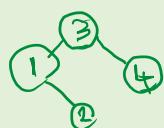
در روش در بدترین حالت زمانی رخ می دهد کلید ها به صورت مرتب وارد شوند (درخت راست نورد تشکیل می شود)

$$\theta(n^2)$$

$$\theta(n \log n)$$

$$\theta(n \log n)$$

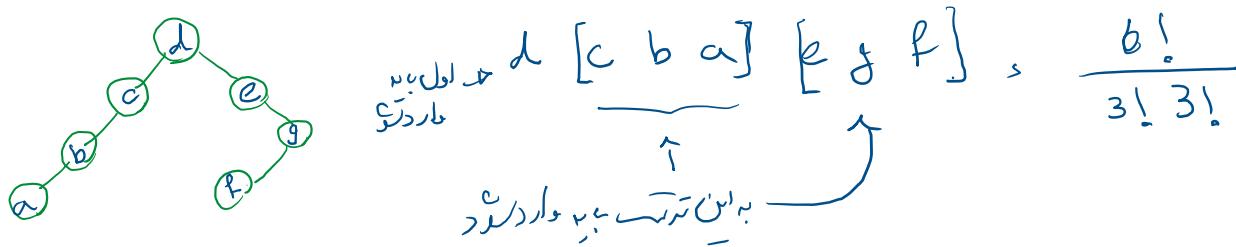
مثال : کلید های ۱, ۲, ۳, ۴ دارای 4 جایگشت اند ، چه تعداد از این جایگشت ها موجب تولید درخت مقابل می شود(روش درج)



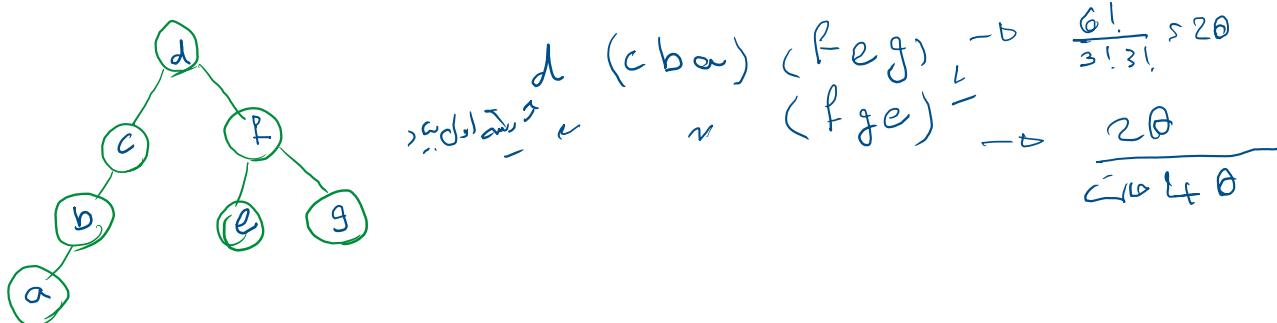
مشخص است اول ۱ وارد شده ، ۱ قبل از ۲ باید وارد شود و ۴ بعد از ۳ باید وارد شود به صورت زیر

$$3 \ 4 \ 1 \ 2, \ 3 \ 1 \ 4 \ 2$$

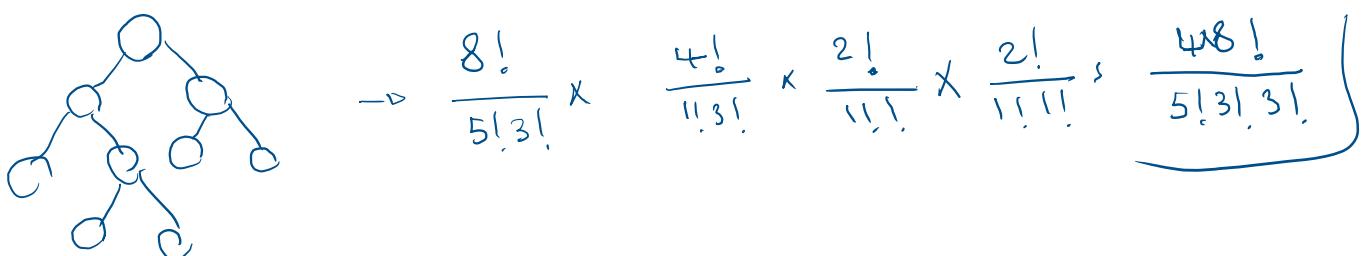
مثال : فرض کنید a < b < c < d < e < f < g این 7 کلید چند جایگشت دارند که درخت مقابل را تولید کند ؟



مثال : فرض کنید $a < b < c < d < e < f < g$ این 7 کلید چند جایگشت دارند که درخت مقابل را تولید کند ؟



مثال : فرض کنید $a < b < c < d < e < f < g$ این 7 کلید چند جایگشت دارند که درخت م مقابل را تولید کند ؟



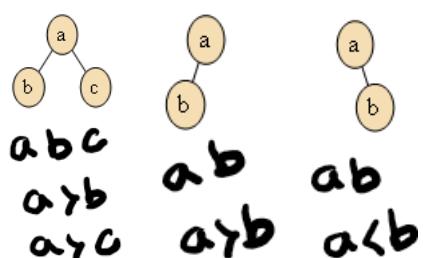
مثال : کدام یک از گزینه های زیر پیمایش پیش ترتیب یک درخت دودویی جست و جو با 12 گره می باشد؟

ب) 12,8,5,10,9,20,14,13,17,15,16,19

الف) 12,8,5,10,9,20,14,15,18,13,16,19

د) 20,14,13,12,8,5,9,10,18,19,15,16

ج) 20,14,13,12,8,5,9,18,10,19,15,16



یک راه رسم هر گزینه و دید که آیا امکان رسم آن گزینه هست یا خیر .
اما میدانیم پیمایش pre order به صورت مقابل است.

شرط لازم و کافی برای آنکه یک دنباله شامل n عدد ، دنباله pre order یک bst باشد آن است که به ازای هر کلید مثل x در دنبال همه اعداد بعد از x از x بزرگتر باشند یا همگی از x کوچکتر باشند یا ابتدا کوچکتر باشند بعد بزرگتر باشند

در نتیجه گزینه ۲ درست است.

مثال : اگر $abcdfbeg$ پیمایش Preorder یک درخت دودویی باشد ، کدام یک از دنباله های زیر نمی تواند پیمایش آن درخت باشد ؟

fdbcagc(د)

fdecbag(ج)

cabfged(ب)

cdbfage(الف)

یکی از راه ها این است که Preorder با هر گزینه را درختش رسم شود .

میتوان تست های مربوط به پیمایش را به bst بربط داد ، و به جای حروف عدد گذاشت ، میدانیم inorder در bst صعودی است یعنی گزینه های را به صورت صعودی بسازیم و در صورت سوال قرار دهیم و ببینم درست است یا خیر.

پیاده سازی BST

: ساختار کلاس و نود های BST

```
#include <iostream>
#include <string>
#include<stack>
using namespace std;
struct BSTNode
{
    int data;
    BSTNode* left;
    BSTNode* right;
    BSTNode* parent;
    BSTNode(int data):data(data)
    {
        left = nullptr;
        right = nullptr;
        parent = nullptr;
    }
};
class BST
{
    BSTNode* root;
    BSTNode* Insert(BSTNode* root , int value);
    void Inorder(BSTNode* root , string& result);
    BSTNode* Search(BSTNode* root, int value);
    BSTNode* Min(BSTNode* root);
    BSTNode* Max(BSTNode* root);
    void Delete(BSTNode* node);
    BSTNode* successor(BSTNode* );
public:
    BST();
    ~BST();
    void Insert(int value);
    BSTNode* Search(int value);
    void Inorder(string& result);
    int Min();
    int Max();
    int successor(int value);
    int predecessor(int value);
    void Delete(int value);
};


```

: سازنده و مخرب کلاس

```
BST::~BST()
{
    delete root;
}
BST::BST()
{
```

افزودن

```

void BST::Insert( int value)
{
    if (!root) root = new BSTNode(value);
    else Insert(root, value);
}
BSTNode* BST::Insert(BSTNode* root, int value)
{
    if (!root)
    {
        root = new BSTNode(value);
        return root;
    }
    if (value > root->data)
    {
        root->right = Insert(root->right, value);
        root->right->parent = root;
    }
    else
    {
        root->left = Insert(root->left, value);
        root->left->parent = root;
    }
    return root;
}

```

جست و جو :

```

BSTNode* BST::Search(BSTNode* root, int value)
{
    if (root != nullptr)
    {
        if (root->data == value) return root;
        if (root->data < value) Search(root->right, value);
        else Search(root->left, value);
    }
    else return nullptr;
}
BSTNode* BST::Search(int value)
{
    return Search(root, value);
}

```

پیدا کردن min و max :

```

BSTNode* BST::Min(BSTNode* root)
{
    BSTNode* tmp = root;
    while (tmp->left != nullptr) tmp = tmp->left;
    return tmp;
}
BSTNode* BST::Max(BSTNode* root)
{
    BSTNode* tmp = root;
    while (tmp->right != nullptr) tmp = tmp->right;
    return tmp;
}
int BST::Min()
{
    return Min(root)->data;
}
int BST::Max()
{
    return Max(root)->data;
}

```

:predecessor و successor پیدا کردن

```

int BST::successor(int value)
{
    BSTNode* tmp = Search(value);
    if (tmp->right) return Min(tmp->right)->data;
    else return tmp->parent->data;
}
int BST::predecessor(int value)
{
    BSTNode* tmp = Search(value);
    if (tmp->left) return Max(tmp->left)->data;
    else return tmp->parent->data;
}

BSTNode* BST::successor(BSTNode* tmp)
{
    if (tmp->right)
        return Min(tmp->right);
    else if (tmp->left)
        return tmp->parent;
    else return tmp;
}
void BST::Delete(BSTNode* node)
{
    if (root != node)
    {
        if (!node->right && !node->left)
        {
            if (node->parent->left == node) node->parent->left = nullptr;
            else node->parent->right = nullptr;
        }
        else if (!node->right && node->left)
        {
            if (node->parent->left == node) node->parent->left = node->left;
            else node->parent->right = node->left;
        }
        else if (node->right && !node->left)
        {
            if (node->parent->left == node) node->parent->left = node->right;
            else node->parent->right = node->right;
        }
        else if (node->right && node->left)
        {
            auto tmp = successor(node);
            tmp->parent = node->parent;
            tmp->right = node->right == tmp ? node->right->right : node->right;
            tmp->left = node->left == tmp ? node->left->left : node->left;
            tmp->right->parent = tmp;
            tmp->left->parent = tmp;
            if (node->parent->left == node) node->parent->left = tmp;
            else node->parent->right = tmp;
        }
        else throw exception();
    }
    else
    {
        if (!node->right && !node->left) root = nullptr;
        else if (!node->right && node->left)
        {
            delete root;
            root = root->left;
        }
        else if (node->right && !node->left)
        {
            delete root;
            root = root->right;
        }
        else if (node->right && node->left)
        {
            auto tmp = successor(node);

```

حذف کردن :

```

        //delete root;
        tmp->parent = node->parent;
        tmp->right = node->right == tmp ? node->right->right : node->right;
        tmp->left = node->left == tmp ? node->left->left : node->left;
        tmp->right->parent = tmp;
        tmp->left->parent = tmp;
        delete root;
        root = tmp;
    }
    else throw exception();
}
}

void BST::Delete(int value)
{
    Delete(Search(value));
}

```

پیمایش in order :

```

void BST::Inorder(string& res)
{
    if (!root) throw exception();
    Inorder(root, res);
}
void BST::Inorder(BSTNode* root, string& res)
{
    if (root == nullptr) return;
    Inorder(root->left, res);
    res += to_string(root->data) + " ";
    Inorder(root->right, res);
}

```

تست کلاس :

```

int main()
{
    BST bst;
    bst.Insert(5);
    bst.Insert(6);
    bst.Insert(2);
    bst.Insert(1);
    bst.Insert(7);
    bst.Insert(9);
    bst.Insert(8);
    bst.Insert(10);
    bst.Insert(11);
    bst.Delete(5);
    string str = "";
    bst.Inorder(str);
    cout << str << endl;
    if (bst.Search(6) != nullptr) cout << 6 << " Found " << endl;
    else cout << 6 << " Not Found " << endl;
    if (bst.Search(0) != nullptr) cout << 0 << " Found " << endl;
    else cout << 0 << " Not Found " << endl;
    cout << "Min = " << bst.Min() << endl;
    cout << "Max = " << bst.Max() << endl;
    cout << "successor(1) = " << bst.successor(1) << endl;
    cout << "predecessor(9) = " << bst.predecessor(6) << endl;
    return 0;
}

```

خروجی :

```

11 10 9 8 7 6 2 1
6Found
0Not Found
Min = 1
Max = 11
successor(1) = 2
predecessor(9) = 2

```

(binomial heap, finocchi heap, deep heap)

این ساختمان داده، برای جست و جو مناسب نیست.

هرم دودویی (Heap)

دونوع هرم دودویی داریم ۱- کمینه ۲- بیشینه

هرم دودویی کمینه (Binary min Heap)

یک درخت دودویی کامل است، که در هر node یک کلید وجود دارد، که کلید هر نود، از کلید فرزندانش کوچکتر یا مساوی است.

هرم دودویی بیشینه (Binary max Heap)

یک درخت دودویی کامل است، که در هر node یک کلید وجود دارد، که کلید هر نود، از کلید فرزندانش بزرگتر یا مساوی است.

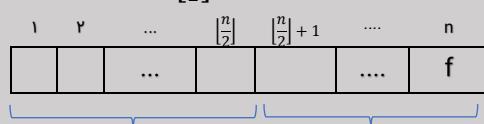
یادآوری :

در نمایش درخت کامل با آرایه داشتیم :

$$\theta(1) \text{ با مرتبه} \begin{cases} parent(i) = \lfloor \frac{i-1}{2} \rfloor & i \neq 0 \\ LeftChild(i) = 2.i + 1 & 2.i + 1 \leq n \\ RightChild(i) = 2.i + 2 & 2.i + 2 \leq n \end{cases}$$

۰	۱	۲	۳	۴	۵
a	b	c	d	e	f

جهت تسهیل فرض کنید آرایه ها از ۱ تا $\lceil \frac{n}{2} \rceil$ شماره گذاری می شوند، همچنین دیدیم که در درخت $\lceil \frac{n}{2} \rceil$ نود داخلی داریم و برگ داریم، یعنی $\lceil \frac{n}{2} \rceil$ تای آخر آرایه برگ هستند.

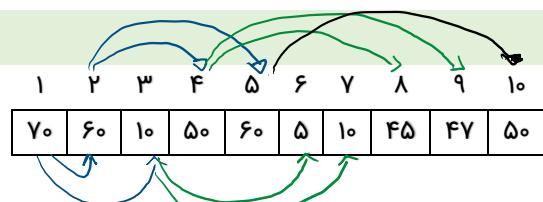


$\left\lfloor \frac{n}{2} \right\rfloor$ $\left\lceil \frac{n}{2} \right\rceil$

در این صورت داریم :

$$\theta(1) \text{ با مرتبه} \begin{cases} parent(i) = \lfloor \frac{i}{2} \rfloor & i \neq 0 \\ LeftChild(i) = 2.i & 2.i \leq n \\ RightChild(i) = 2.i + 1 & 2.i + 1 \leq n \end{cases}$$

مثال : آیا آرایه زیر max heap می باشد یا خیر ؟



Child های هر parent را مشخص میکنیم، اگر child های هر

parent از خود آن بزرگتر یا مساوی باشد، هرم دودویی، بیشینه است.

که در این آرایه این شرط برقرار است و درخت بیشینه می باشد، همچنین میتوان شکل درختی آن را رسم کرد.

تشخیص اینکه یک آرایه n تایی هرم بیشینه یا کمینه هست یا نیست از مرتبه $(n)0$ است.(برای هرم بیشینه داریم :

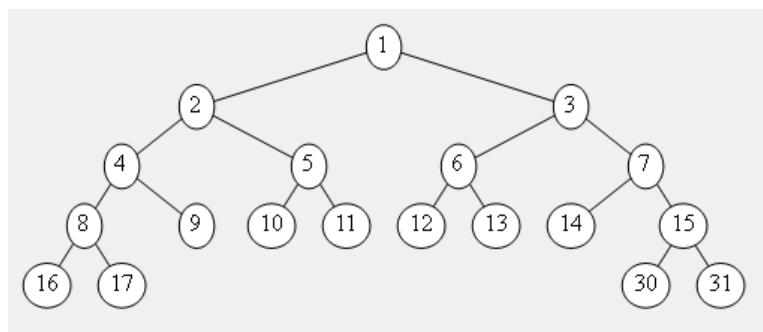
```
A[0...n]
for(i = 0 to n/2]
    if(A[i]<A[2i+1] || A[i] < A[2i+2])
        return false;
return true;
```

در هرم بیشینه عنصر بیشینه و کمینه کجا هستند و مرتبه زمانی یافتن آنها را بدست آورد. آرایه به صورت : $A[1...n]$ و کلید ها متایز می باشد.

در هرم بیشینه ، عنصر بیشینه همان ریشه می باشد که مرتبه زمانی پیدا کردن آن $(1)\theta$ می باشد ، در حالی که عنصر کمینه در یکی از برگ ها قرار دارد ، یعنی باید $\left[\frac{n}{2}\right]$ پایانی آرایه را بررسی کنیم ، که لازم است $1 - \left[\frac{n}{2}\right]$ لازم است ، که مرتبه $(n)\theta$ است . برعکس این شرایط برای هرم کمینه نیز وجود دارد یعنی در هرم کمینه یافتن کمینه $(1)\theta$ است و یافتن بیشینه $(n)\theta$ است.

مثال : در هرم کمینه با تعداد بیشمار کلید ، اولین و دومین و سومین و چهارمین و به طور کلی k ام کمینه در کدام خانه می باشند.(کلید ها متایز است)

فرض کنید کلید های به صورت $1, 2, 3, 4, \dots$ است داریم :



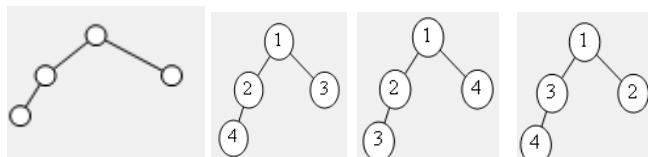
اولین \min خانه $[1]$

دومین \min در خانه ۲ یا ۳ می توان باشد چون در این دو خانه هستن که فقط یک عنصر کوچکتر (عنصر خانه اول) وجود دارد ، مثلاً در خانه ۴ نمیتواند باشد چون در این صورت ۲ خانه باید از ان کوچکتر باشد درحالی که خود دومین عنصر کوچک است.

سومین عنصر کمینه میتواند در خانه های ۲ یا ۳ یا ۴ یا ... نمیتواند باشد اما ۸ و ۹ و ... نمیتواند باشد چون ۸ سه جد دارد چهارمین \min می تواند از ۲ تا ۱۵ می تواند باشد.

K امین کمینه برای $i=1$ می تواند در نود های ۲ تا $1 - 2^k$ قرار گیرد.

مثال : با ۴ کلید متایز چند هرم کمینه می تواند ساخت؟

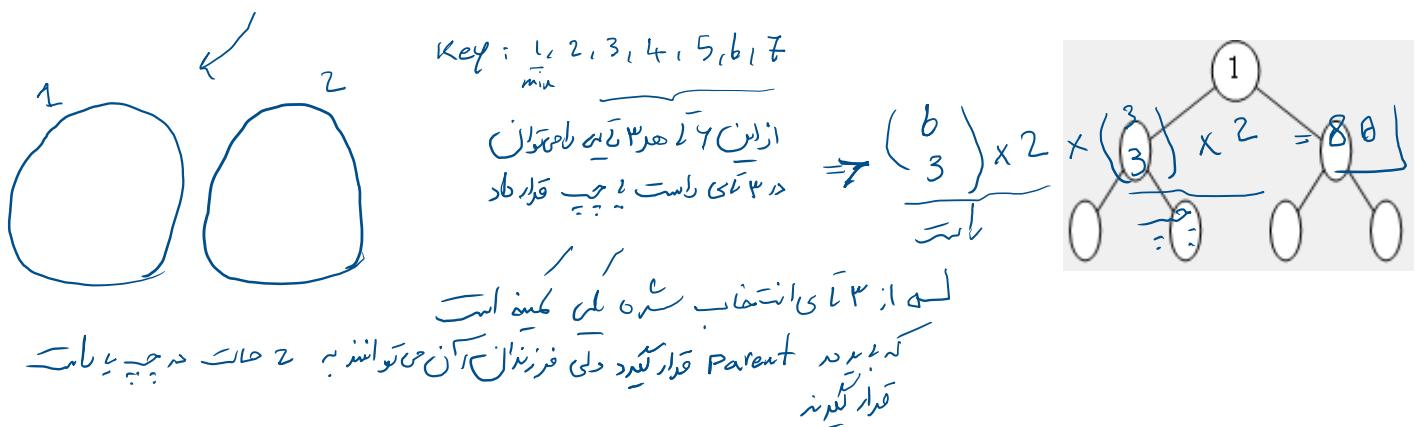


از نظر شکل (توپولوژی) درخت باید کامل باشد که فقط یک حالت می باشد .

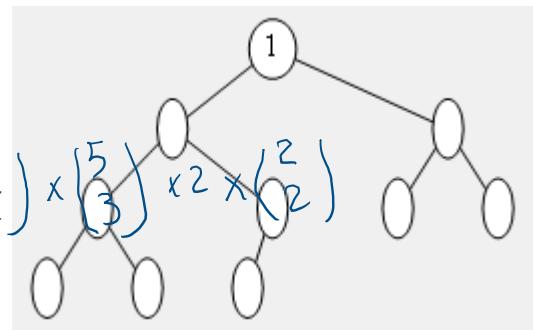
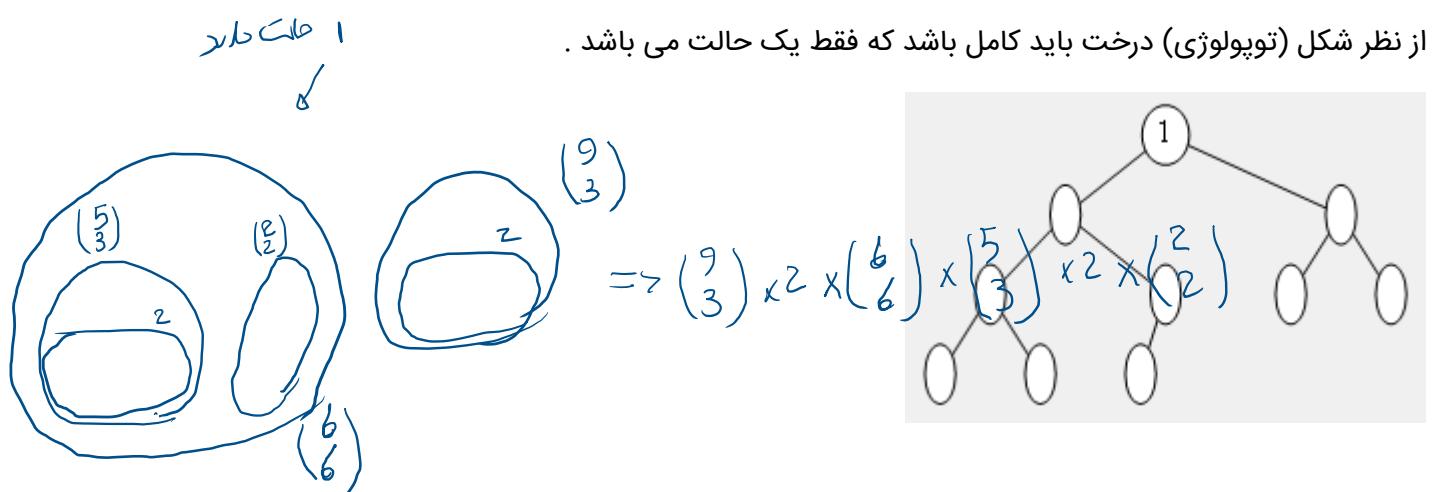
مثال : با ۷ کلید متایز چند هرم کمینه می تواند ساخت؟

از نظر شکل (توپولوژی) درخت باید کامل باشد که فقط یک حالت می باشد .

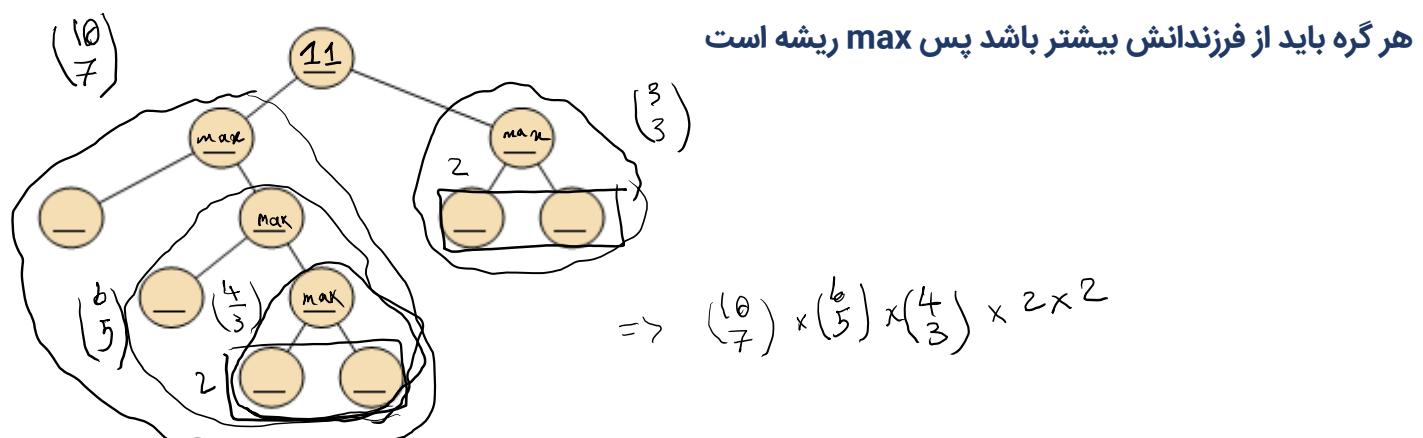
فقط این حالت \min است



مثال : با 10 کلید متمایز چند هرم کمینه می توانند ساخت؟



مثال : به چند حالت می توان اعداد 1 تا 11 را در گره های درخت زیر قرار داد به طوری که هر گره از فرزندانش بزرگتر باشد.



درج در هرم بیشینه

الگوریتم های برای هرم بیشینه گفته می شود ، برای هرم کمینه نیز مشابه همین می باشد.

برای درج توجه کنید که درخت کامل است یعنی از چپ باید اولین جای خالی، برای `insert` است، بعد از `insert` کردن عنصر را با اجدادش آنقدر مقایسه میکنیم تا هرم به حالت درست خود برسد.

مرتبه درج $O(\log n)$ می باشد.

مثال : به ترتیب کلید های 40 و 50 را در هرم بیشینه زیر درج کنید.

A	1	2	3	4	5	6	7	8
	۴۵	۳۰	۲۵	۲۷	۲۰	۲۲		

در آرایه بالا که ایستاد است (طول آرایه ثابت است) $\text{heapSize}(A) = 6$ و $\text{length}(A) = 8$ می باشد.

برای درج 40 به صورت زیر داریم

A	1	2	3	4	5	6	7	8
	۴۵	۳۰	۲۵	۲۷	۲۰	۲۲	۴۰	

حال 40 را با `parent` اش یعنی 25 مقایسه میکنیم، از خود بیشتر است پس تعویض می شود :

A	1	2	3	4	5	6	7	8
	۴۵	۳۰	۴۰	۲۷	۲۰	۲۲	۲۵	

حال 40 را با `parent` جدیداً مقایسه می کنیم یعنی 1 کمی کوچکتر است و در جای خود می ماند ، حال برای درج 50

داریم : $\text{heapSize}(A) = 8$

A	1	2	3	4	5	6	7	8
	۴۵	۳۰	۴۰	۲۷	۲۰	۲۲	۲۵	۵۰

حال 50 را با `parent` خود یعنی 4 مقایسه میکنیم که باید جا به جا شود

A	1	2	3	4	5	6	7	8
	۴۵	۳۰	۴۰	۵۰	۲۰	۲۲	۲۵	۲۷

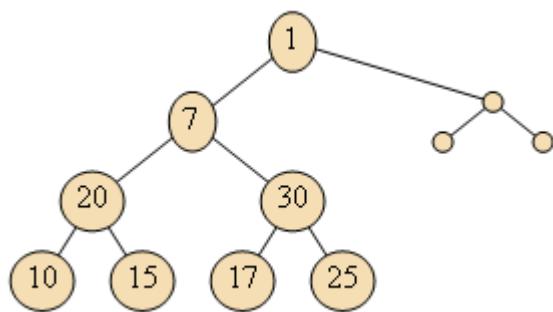
سپس با عنصر 2 ام مقایسه میکنیم که باید جا به جا شود

A	1	2	3	4	5	6	7	8
	۴۵	۵۰	۴۰	۳۰	۲۰	۲۲	۲۵	۲۷

حال با عنصر اول مقایسه می کنیم :

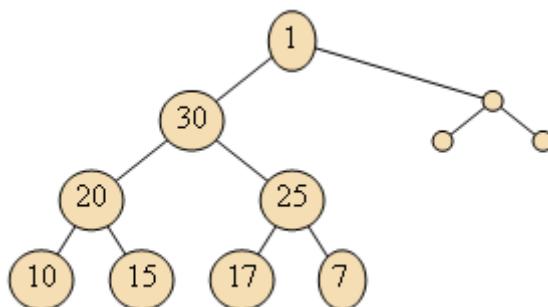
A	1	2	3	4	5	6	7	8
	۵۰	۴۵	۴۰	۳۰	۲۰	۲۲	۲۵	۲۷

عمل maxheapify



برای i maxheapify داریم ، مثلاً درخت مقابل را در نظر بگیرید عدد 7 نظرم maxheap را بهم ریخته است عمل maxheapify برای این عنصر یعنی عدد 7 را با فرزندانش مقایسه کنیم(مشروط بر اینکه عناصر چپ و راست خودشان هرم بیشینه باشند) و با فرزند بزرگتر جا به جا کنیم سپس این کار را به صورت بازگشتی ادامه دهیم.

اگر این عمل را انجام دهیم به درخت زیر می‌رسیم.



عمل کاهش کلید یک هرم بیشینه به این صورت است که یک عنصر مشخص مثل $A[i]$ به ما داده شده است ، و از ما خواسته اند که مقدار این کلید را کاهش دهیم (مثلاً اگر 50 است به 7 تغییرش دهیم) ، مرتبه زمانی این عملیات $O\left(\log\left(\frac{n}{i}\right)\right) = O(\log n)$ می‌باشد .

ساخت هرم بیشینه

N کلید به ما داده شده است و از ما میخواهند هرم بیشینه بسازیم :

۱- اگر کلید ها یکی داده شوند n بار عمل درج انجام می شود.

مرتبه حالت بالا $O(n \log n)$ می‌باشد.

۲- اگر همه عناصر یکجا داده شده باشند ، می‌توان با کمک maxheapify هرم بیشینه ساخت ، از آخرین گره داخلی(گره ای که فرزندانش برگ اند) به سمت ریشه حرکت میکنیم و عمل maxheapify را انجام می‌دهیم.

```
A[1...n]
for(i = n/2 downto 1)
    maxheapify(i)
```

مرتبه حالت بالا $\theta(n)$ می‌باشد.

هرم های ساخته شده در دو روش فوق لزوماً یک شکل نیستند.

مثال : ثابت کنید ساخت هرم با n عنصر داده شده از مرتبه $(n) \theta$ است.

در هرم n تایی (یعنی یک درخت کامل n تایی) حداقل تعداد $\left\lceil \frac{n}{2^{h+1}} \right\rceil$ نод به ارتفاع h وجود دارد.

$$h(0) = \left\lceil \frac{10}{2^{0+1}} \right\rceil = 5, h(1) = \left\lceil \frac{10}{2^{1+1}} \right\rceil = 3, h(2) = \left\lceil \frac{10}{2^{2+1}} \right\rceil = 2, \dots$$

مشخص است که ما حداقل 5 نod داریم به ارتفاع 0 و حداقل 3 نod داریم به ارتفاع 1 و (به صورت استقرائی قابل اثبات است و برای درخت پر دقیقاً $\left\lceil \frac{n}{2^{h+1}} \right\rceil$ نod به ارتفاع h وجود دارد.)

همچنین مشخص است Heapify اگر رویه نod x اجرا شود زمانش به ارتفاع x بستگی دارد (زیر به عناصر پایین آن عنصر وابسته است) ، بهتر بگوییم زمان heapify برابر است با ارتفاع h نod که ما حداقل $\left\lceil \frac{n}{2^{h+1}} \right\rceil$ نod به ارتفاع h داریم پس زمان اجرای رویه همه نod های داخلی برابر است با :

$$\sum_{h=1}^{\lfloor \log n \rfloor} h \cdot \left\lceil \frac{n}{2^{h+1}} \right\rceil \cong n \cdot \sum_{h=1}^{\log n} \frac{h}{2^{h+1}} \text{ سیگما عدد است } \theta(n)$$

مثال :

n عدد در آرایه T ذخیره شده اند. برای T ، الگوریتم ذکر شده را اجرا می کنیم. اگر t تعداد دفعات اجرای حلقه repeat باشد، کدام یک از گزاره های زیر صحیح است با فرض این که $.k=[\lg n]$

```

Procedure M(T[1..n])
for i ← ⌊ n / 2 ⌋ downto 1 do
    R ← i
    Repeat
        j ← R
        if 2j ≤ n and T[2j] > T[R] then
            R ← 2j
        if 2j+1 ≤ n and T[2j+1] > T[R] then
            R ← 2j+1
        swap T[j], T[R]
    until j = R

```

$$t \leq 2 \times 2^{k-2} + 3 \times 2^{k-3} + \dots + k \quad (1)$$

$$t \leq 3 \times 2^{k-2} + 4 \times 2^{k-3} + \dots + k \quad (2)$$

$$t \leq 3 \times 2^{k-2} + 4 \times 2^{k-3} + \dots + (k+1) \quad (3)$$

$$t \leq 2 \times 2^{k-1} + 3 \times 2^{k-2} + \dots + (k+1) \quad (4)$$

راهنمایی : کد فوق داخل repeat عمل ، heapify انجام می شود

پاسخ : گزینه 4

حذف ماقزیم از هرم بیشینه (heap extractmax)

ماکسیم در ریشه است که برای حذف کافی است آن را با آخرین عنصر در هرم تعویض کنیم و `heapsize` را یک واحد کم کنیم و سپس رویه عنصر ریشه `heapify` کنیم.

مثال : دو بار عمل حذف ماقزیم را در هرم زیر نشان دهید.

۱	۲	۳	۴	۵	۶	۷	۸
۵۰	۴۵	۲۰	۴۰	۳۰	۱۰	۱۵	۳۵

حذف اول ۷ وقتی `heapsize` کم میشود از ۱ تا ۷ جزو `heap` حساب می شود.

۱	۲	۳	۴	۵	۶	۷	۸
۳۵	۴۵	۲۰	۴۰	۳۰	۱۰	۱۵	۵۰

حال برای عنصر اول `heapify` میکنیم و به آرایه زیر میرسیم.

۱	۲	۳	۴	۵	۶	۷	۸
۴۵	۴۰	۲۰	۳۵	۳۰	۱۰	۱۵	۵۰

حذف دوم :

۱	۲	۳	۴	۵	۶	۷	۸
۱۵	۴۰	۲۰	۳۵	۳۰	۱۰	۴۵	۵۰

حال برای عنصر اول `heapify` را اعمال میکنیم تا به آرایه زیر برسیم.

۱	۲	۳	۴	۵	۶	۷	۸
۴۰	۳۵	۲۰	۱۵	۳۰	۱۰	۴۵	۵۰

حال برای حذف سوم بعد از `heapify` داریم :

۱	۲	۳	۴	۵	۶	۷	۸
۱۰	۳۰	۲۰	۱۵	۱۰	۴۰	۴۵	۵۰

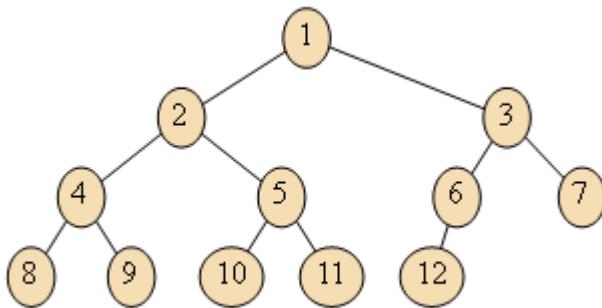
اگر توجه کنید عناصری که حذف می شوند `sort` شده اند و اگر آخرین عنصر را حذف کنیم به آرایه مرتب می رسیم.

روش مرتب سازی هرمی

ابتدا با اعداد داده شده یک هرم بیشینه می سازیم سپس ۷ بار عمل حذف ریشه انجام میدهیم

ادامه این بخش در درس الگوریتم توضیح داده خواهد شد.

مثال : عنصر $A[i]$ از هرم بیشنه را حذف کنید ، چگونه انجام می دهد.



فرض کنید می خواهیم عنصر $A[i] = A[4]$ را حذف کنیم برای این کار ، آخرین عنصر را با آن جایگزین میکنیم و یکی از کم میکنیم ، مقدار جدید آ را با پدرس مقایسه می دهیم اگر از پدرس بزرگتر بود رو به بالا عمل تعویض را انجام می دهیم اگر اینطور نبود با فرزندانش مقایسه کنید اگر حداقل از یکی از فرزندانش کمتر بود عمل heapify را انجام دهیم.

مرتبه عمل حذف $O(\log n)$ می باشد.

مثال : ادغام دو هرم بیشنه m و n عنصری از چه مرتبه ای است ؟

برای ادغام دو هرم بیخیال هرم بودن شوید (چه هردو کمینه باشند چه یکی کمینه یکی بیشنه و چه هردو بیشنه) ، شما دو آرایه دارید که میخواهید با آن هرم بسازید ، مرتبه ساخت هرم وقتی عناصر را داشتیم از مرتبه $\theta(n)$ بود یعنی در اینجا از مرتبه $\theta(n+m)$ است.

صف اولویت (Priority Queue)

در این صف هر عنصر یک کلید دارد که این کلید اولویت نامیده می شود و همیشه عنصری زود تر سرویس می گیرد و از صف خارج می شود که اولویت آن بیشتر است. برای اولویت بالاتر میتوان مثلًا گفت آنی که مقدار کلیدش بزرگ تر است

ساخت صف اولویت

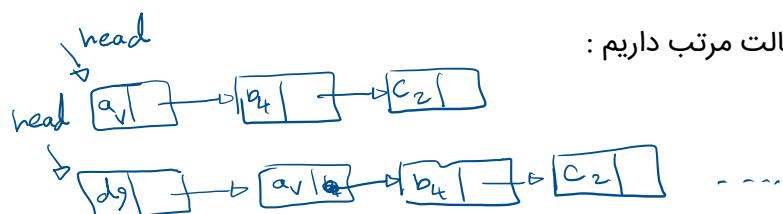
برای ساخت صف اولویت از لیست پیوندی استفاده می کنیم که دو حالت داریم :

- ۱- لیست پیوندی مرتب
- ۲- لیست پیوندی نامرتب

فرض کنید میخواهیم جدول مقابل را از بالا به پایین درج کنیم (کلید بزرگتر اولویت بیشتری دارد)

عنصر	اولویت
a	۷
b	۴
c	۲
d	۹
e	۶
f	۳

برای حالت مرتب داریم :



سرچ میکنیم و جای هر عنصر را پیدا مکنیم ، مزیتی که این روش دارد اولین عنصر بیشترین اولویت را دارد .

در این روش عمل درج $O(n)$ و عمل حذف $O(1)$ است .

در روش نامرتب هر عنصر را اول درج میکنیم موقع حذف سرج میکنیم که اولویت بیشتر از آن نباشد.

در روش های گفته شده ، روشی بهتر نیست ، هردو معایت و فوایدی دارند ، اما بهترین روش ساخت صف اولیوت هرم است، کافی است بر اساس اولویت ها هرم بیشینه یا کمینه بسازید و از روش های ساخت هرم استفاده کنید ، در این حالت ریشه بیشترین اولویت دارد و عمل درج و حذف مانند هرم انجام می شود که از مرتبه $O(\log n)$ می باشد.

پیاده سازی هرم کمینه

ساختار کلاس :

```
#include <iostream>
using namespace std;
class MinHeap
{
private:
    int* heap;
    int arraysize;
    int heapsize;
    inline int Parent(int index);
    inline int LeftChild(int index);
    inline int RightChild(int index);
public:
    MinHeap(int arraysize);
    ~MinHeap();
    void Insert(int key);
    void MinHeapify(int index);
    int ExtractMin();
    int Delete(int index);
    void DecreaseKey(int i, int newvalue);
    int GetMin();
    static MinHeap* Merge(MinHeap* heap1, MinHeap* heap2);
};
```

سازنده و مخرب کلاس :

```
MinHeap::MinHeap(int arraysize) : arraysize(arraysize), heapsize(0)
{
    heap = new int[arraysize];
    memset(heap, 0, arraysize);
}
MinHeap::~MinHeap()
{
    delete heap;
}
```

بدست آوردن child و parent ها :

```
int MinHeap::Parent(int index)
{
    return (index - 1) / 2;
}
int MinHeap::LeftChild(int index)
{
    return 2 * index + 1 < heapsize ? 2 * index + 1 : -1;
}
int MinHeap::RightChild(int index)
{
    return 2 * index + 2 < heapsize ? 2 * index + 2 : -1;
}
```

افزودن عنصر

```
void MinHeap::Insert(int key)
{
    if (heapsize < arraysize)
    {
        heap[heapsize] = key;
        int current = heapsize;
        int parent = Parent(current);
        while (parent != -1 && heap[current] < heap[parent] && current > 0)
        {
            swap(heap[current], heap[parent]);
```

```

        current = parent;
        parent = Parent(current);
    }
    heapsize++;
}
else throw exception();
}

```

بدست آورد

```

void MinHeap::MinHeapify(int index)
{
    if (heapsize <= 0) throw exception();
    if (heapsize > 1)
    {
        int l = LeftChild(index);
        int r = RightChild(index);
        int smallest = index;
        if (l != -1 && heap[l] < heap[index])
            smallest = l;
        if (r != -1 && heap[r] < heap[smallest])
            smallest = r;
        if (smallest != index)
        {
            swap(heap[index], heap[smallest]);
            MinHeapify(smallest);
        }
    }
}

```

ExtractMin

```

int MinHeap::ExtractMin()
{
    int res;
    if (heapsize <= 0)
        throw exception();
    else if (heapsize == 1)
        res = heap[heapsize--];
    else
    {
        res = heap[0];
        swap(heap[0], heap[heapsize-1]);
        heapsize--;
        MinHeapify(0);
    }
    return res;
}

```

حذف :

```

int MinHeap::Delete(int index)
{
    int res;
    if (heapsize <= 0)
        throw exception();
    else if (heapsize == 1)
        res = heap[heapsize--];
    else
    {
        res = heap[index];
        swap(heap[index], heap[heapsize]);
        heapsize--;
        MinHeapify(index);
        while (Parent(index) != -1 && heap[index] < heap[Parent(index)])
        {
            swap(heap[index], heap[Parent(index)]);
            index = Parent(index);
        }
    }
    return res;
}

```

کاهش کلید :

```
void MinHeap::DecreaseKey(int i, int newvalue)
{
    heap[i] = newvalue;
    while (Parent(i) != -1 && heap[Parent(i)] > heap[i])
    {
        swap(heap[i], heap[Parent(i)]);
        i = Parent(i);
    }
}
```

: min مشاهده

```
int MinHeap::GetMin()
{
    return heap[0];
}
```

ادغام دو هیپ :

```
MinHeap* MinHeap::Merge(MinHeap* heap1, MinHeap* heap2)
{
    MinHeap* resheap = new MinHeap(heap1->arraysize + heap2->arraysize);
    for (int i = 0; i < heap1->heapsize; i++)
        resheap->heap[i] = heap1->heap[i];
    for (int i = 0; i < heap2->heapsize; i++)
        resheap->heap[heap1->heapsize + i] = heap2->heap[i];
    resheap->heapsize = heap1->heapsize + heap2->heapsize;

    for (int i = (resheap->heapsize)/2; i >= 0; i--)
        resheap->MinHeapify(i);

    return resheap;
}
```

تست :

```
int main()
{
    MinHeap heap1(10);
    heap1.Insert(3);
    heap1.Insert(2);
    heap1.Insert(1);
    heap1.Insert(15);
    heap1.Insert(5);
    heap1.Insert(4);
    heap1.Insert(45);
    cout << heap1.ExtractMin() << " ";
    cout << heap1.ExtractMin() << " ";
    heap1.DecreaseKey(3, -2);
    cout << heap1.GetMin() << " ";
    MinHeap heap2(5);
    heap2.Insert(1);
    heap2.Insert(5);
    heap2.Insert(-10);
    heap2.Insert(2);
    MinHeap heapMerge = *MinHeap::Merge(&heap1, &heap2);
    cout << heapMerge.ExtractMin() << " ";
    cout << heapMerge.ExtractMin() << " ";
    return 0;
}
```

خروجی :

1 2 -2 -10 -2

کوییز ها

کوییز شماره ۱

روابط بازگشته زیر را بدست آورید.

$$T(n) = 2T(n-1) + 3^n, T(1) = 1, T(0) = 0$$

حل:

معادله مشخصه رابطه به صورت زیر می باشد.

$$(x-2)(x-3) = 0 \rightarrow x_1 = 2, x_2 = 3$$

داریم:

$$T(n) = \alpha_1 2^n + \alpha_2 3^n$$

برای $T(0) = 0$ و $T(1) = 1$ داریم:

$$T(0) = \alpha_1 + \alpha_2 = 0 \rightarrow \alpha_2 = -\alpha_1$$

$$T(1) = 2\alpha_1 + 3\alpha_2 = 1 \rightarrow 2\alpha_1 - 3\alpha_1 = -\alpha_1 = 1 \rightarrow \alpha_1 = -1, \alpha_2 = 1$$

بنابراین:

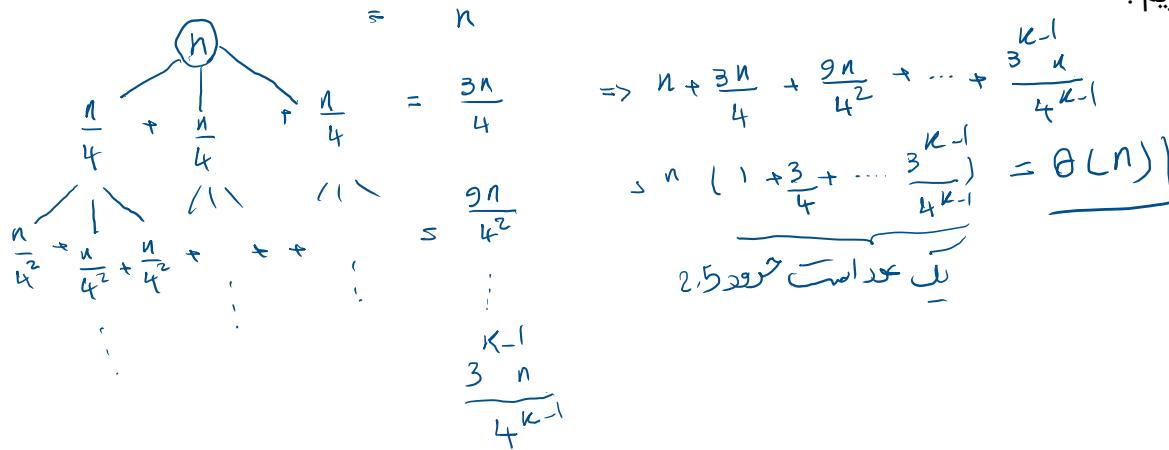
$$T(n) = -2^n + 3^n$$

$$T(n) = 3T\left(\frac{n}{4}\right) + n$$

باتوجه به قضیه مستر داریم:

$$n \in \Omega(n^{\log_4 3 + \varepsilon}) \rightarrow T(n) = \theta(n)$$

از درخت کامل داریم:



$$T(n) = 16T\left(\frac{n}{2}\right) + n^3 \log n$$

باتوجه به قضیه مستر داریم :

$$n^3 \log n \in O(n^{\log_2 16 - \varepsilon}) \rightarrow T(n) = \theta(n^4)$$

$$T(n) = 9T\left(\frac{n}{2}\right) + n^2 \log n$$

باتوجه به قضیه مستر داریم :

$$n^2 \log n \in O(n^{\log_2 9 + \varepsilon}) \rightarrow T(n) = \theta(n^{\log_2 9})$$

کوییز شماره ۲

الگوریتم جستجوی دو دویی را به گونه ای تغییر دهید که بجای تقسیم آرایه به دو زیر آرایه آن را به ۳ زیر آرایه تقسیم کرده و جستجو را نجام دهد. الگوریتم را تحلیل کرده و پیچیدگی زمانی آنرا بدست آوردید

```
int BinarySearch(int l , int h , int key,int A[])
{
    if(l<=h)
    {
        int m1 = (l+h)/3;
        int m2 = 2*(l+h)/3;
        if(A[m1]<key)
            BinarySearch(l,m1-1,key,A);
        else if (A[m2]<key)
            BinarySearch(m1+1,m2-1,key,A);
        else if (A[m2]>key)
            BinarySearch(m2+1,h,key,A);
        else if(A[m1]==key)
            return m1;
        else if (A[m2]==key)
            return m2;
        else if (A[m3]==key)
            return m3;
    }
}
```

آرایه را به ۳ بخش از $\frac{1}{3}$ تا $\frac{2}{3}$ و از $\frac{2}{3}$ تا آخر تقسیم بندی میکنیم و برای هر بخش جست و جوی دودویی را صدا میزنیم :

$$T(n) = 3T\left(\left\lfloor \frac{n}{3} \right\rfloor\right) + \theta(1), T(1) = T(2) = 1$$

$$T(n) = 3T\left(\left\lfloor \frac{n}{3} \right\rfloor\right) + \theta(1) = \theta(n)$$

پیچیدگی زمانی توابع زیر را محاسبه کنید.

$$T(n) = 3n^3 + 2n\sqrt{n} + 2 \log(n!)$$

$$T(n) = 65T\left(\frac{n}{4}\right) + n^3$$

$$T(n) = 3n^3 + 2n\sqrt{n} + 2 \log(n!) = 3n^3 + 2n\sqrt{n} + 2n \cdot \log(n) = \theta(n^3)$$

برای $T(n) = 65T\left(\frac{n}{4}\right) + n^3$ بنا بر مستر داریم :

$$n^3 = O\left(n^{\frac{1}{2}\log 65 + \varepsilon}\right) \rightarrow T(n) = \theta(n^{\frac{1}{2}\log 65})$$

با استفاده از روش تکرار با جایگذاری پیچیدگی زمانی تابع زیر را محاسبه کنید $T(1) = 0$

$$\begin{aligned} T(n) &= 3T\left(\frac{n}{3}\right) + n = 3\left(3T\left(\frac{n}{9}\right) + \frac{n}{3}\right) + n = 3^2T\left(\frac{n}{3^2}\right) + \frac{n}{3} + n = 3^2(3T\left(\frac{n}{3^3}\right) + \frac{n}{3}) + n + n \\ &= 3^3T\left(\frac{n}{3^3}\right) + n + n + n = \dots = 3^kT\left(\frac{n}{3^k}\right) + kn \end{aligned}$$

اگر $3^k = n \rightarrow k = \log n$ باشد آنگاه :

$$T(n) = 3^kT(1) + kn = kn = \theta(n \cdot \log n)$$

کوییز شماره ۳

ماتریس پراکنده زیر را در نظر بگیرید. با استفاده از سریعترین الگوریتم ارائه شده در کلاس(`Terms + Columns`)0 ترانهاده این ماتریس را محاسبه کنید

توضیح: ماتریس اولیه 6×6 میباشد.

توضیح : ابتدا آرایه های `RowSize` و `RowStart` را محاسبه کنید.

	row	col	value
[0]	0	0	12
[1]	0	4	24
[2]	1	3	15
[3]	2	0	-5
[4]	2	3	49
[5]	3	2	37
[6]	4	1	44
[7]	4	2	51

RowIndex 0 1 2 3 4 5

RowSize = [2,1,2,2,1,0]

RowStart =[0,2,3,5,7,8]

هر خونه از خونه قبلی خودش با نظیر خونه قبلی در `rowsize` بدست می آید مثلًا خونه سوم

مجموع ۱ از `rowstart` و ۲ از `rowsize` هست (یا مجموع همه خونه های قبلی در `rowsize` یعنی $1 + 2$)

توضیح : حال برای محاسبه ترانهاده از اول شروع به پیمایش میکنیم و به هر ردیفی که رسیدیم در rowstart مشخص شده آن در خانه ماتریس ترانهاده مقدار آن را جای گذاری میکنیم و یکی به آن ردیف می افزایم یعنی مثلا وقتی خانه $_{5,0}$ را ترا نهاده کردیم و در $_{0,0}$ گذاشتیم در این حال $\text{RowStart} = [1, 2, 3, 5, 7, 8]$ می شود یا وقتی که $_{0,2}$ را قرار دایدم $\text{RowStart} = [2, 3, 5, 5, 7, 8]$ می شود و وقتی $_{3,2}$ را قرار میدهیم $\text{RowStart} = [2, 3, 5, 7, 8]$ و به این صورت برای سایر خانه ها ، پس خواهیم داشت

	row	col	value
[0]	0	0	12
[1]	0	0	2
[2]	1	4	44
[3]	2	3	37
[4]	2	4	51
[5]	3	1	15
[6]	3	2	49
[7]	4	0	24

عبارت prefix و postfix عبارت میان وندی زیر را محاسبه کنید.

$$a/b - c + d * e * c - a$$

برای بدست آوردن pre fix داریم :

$$((a/b) - c) + ((d * e) * c) - a$$

حال کافی است عملگر هارا به قبل از پرانتز های مربوطه منتقل کنیم :

$$-(+ \{- / (ab) c\} * [* (de) c] a)$$

پرانتز هارا پاک میکنیم

برای post fix نیز به همین شکل داریم

$$ab/c - de * c * + a -$$

کوییز شماره ۳

تابعی (توابعی) بنویسید که اشاره گر به ابتدای یک لیست پیوندی دو طرفه بگیرد و آنرا به صورت صعودی مرتب کند. (در لیست پیوندی تنها عدد ذخیره شده است.)

Link List Nodes

```
Template<T = int>
struct Node {
    int data;
    Node* Previous;
    Node* Next;
};

swap

Template<T = int>
void Swap(T & a , T & b)
{
    a = a + b;
    b = a - b;
    a = a - b;
}

BubbleSort
void BubbleSort(Node *head)
{
    if (head == nullptr) return;

    Node *ptr;
    Node *boundarySorted= nullptr; مرز بخش مرتب شده را نگه داری می‌کند

    bool ISswap = true;
    while (ISswap)
    {
        اگر در یک دور پیمایش هیچ جایی جایی انجام نشود یعنی آرایه مرتب است
        ISswap = false;
        ptr = head;

        while (ptr ->Next != boundarySorted) فقط تا جایی که مرتب نشده جلو میرویم
        {
            if (ptr ->data > ptr ->Next->data) فقط مقاییر جایه جایی شود
            {
                Swap(ptr ->Next->data , ptr ->data);
                ISswap = true;
            }
            ptr = ptr ->Next;
        }

        ثبت مرز مرتب بخش مرتب شده
        boundarySorted = ptr;
    }
}
```

تابعی بنویسید که لیست پیوندی دو طرفه بگیرد و آنرا بر عکس نماید

Link List Nodes

```
struct Node {
    int data;
    Node* Previous;
    Node* Next;
};

Reverse
void Reverse(Node *head)
{
    Node *currentNode = head;
    Node *temp = nullptr;
    while (current != nullptr)
    {
        temp = currentNode ->Previous;
        currentNode ->Previous = currentNode ->Next;
        currentNode ->Next = temp;
        currentNode = currentNode ->Previous;
    }
    if(temp != nullptr )
        *head = temp->Previous;
}
```

آزمون پایانی نیم سال دوم ۹۹

پاسخ این آزمون به زودی داده می شود.

۱. ماتریس پراکنده زیر را در نظر بگیرید. با استفاده از سریع ترین الگوریتم ارائه شده در کلاس `(که O(Terms + Columns))` پیچیدگی زمانی آن است) ترانهاده این ماتریس را محاسبه کنید (۲.۵ نمره)

توضیح: ماتریس اولیه 6×6 می باشد.

توضیح ۲: ابتدا آرایه های `RowSize` و `RowStart` را محاسبه کنید.

	row	col	value
[0]	0	0	14
[1]	0	4	22
[2]	1	3	8
[3]	2	0	-2
[4]	2	3	94
[5]	3	2	27
[6]	4	1	44
[7]	4	2	51

۲. پیمایش میان ترتیب و پیش ترتیب یک درخت به صورت زیر است. ابتدا درخت دودویی عبارت را بکشید. سپس پیمایش پیش ترتیب آنرا محاسبه نمایید (۲ نمره)

preorder: $L\ B\ D\ C\ F\ G\ E\ K\ M$

inorder: $C\ D\ B\ G\ F\ L\ E\ M\ K$

postorder?:

۳. الف) پیچیدگی زمانی تابع زیر را فقط بنویسید (۲ نمره)

$$\begin{aligned} 1. \quad T(n) &= 8T\left(\frac{n}{2}\right) + n^3 \\ 2. \quad 2n^23^n + 100n^32^n \end{aligned}$$

$$\begin{aligned} 3. \quad \sum_{i=0}^n i^3 \\ 4. \quad \frac{5n^4}{\log n} + 3n^3 \log n \end{aligned}$$

- ب) پیچیدگی تابع بازگشتی زیر را محاسبه کنید. (۱.۵ نمره)

$$T(n) = T(n - 1) + 3^n$$

- ج) پیچیدگی هر کدام از الگوریتم های زیر به چه صورت است (در حداقل یک خط توضیح دهید) (۱.۵ نمره)

A. برعکس کردن لیست پیوندی دو طرفه

B. درج یک عنصر در یک لیست پیوندی یک طرفه پس از گره X (از نوع گره است).

مروری بر ساختمان‌های داده

۴. ایده اصلی جستجوی **interpolation** چیست و در چه موقعی از آن استفاده می‌کنیم؟ فرمول جستجوی **interpolation** را به گونه‌ای تغییر دهید که تا حدی بتواند در جستجوی کلمات نیز کارایی داشته باشد؟ (۲ نمره)

۵. می‌خواهیم مسائل زیر را حل نماییم. چه ساختمان داده (یا ساختمان داده‌هایی) برای حل این مسائل پیشنهاد می‌دهید و چرا؟ همچنین برای حل هر مورد یک الگوریتم اولیه ارائه دهید: (۷.۵ نمره)

* نیازی به ارائه الگوریتم جزئی نیست. ارائه شبیه کد کلی کافیست. استفاده از الگوریتم‌های مربوط به ساختمان داده‌های پایه ارائه شده در کلاس بلامانع است. به عبارت دیگر می‌توانید یک صف بسازید و از توابع **enqueue** و **dequeue** آن استفاده کنید و نیازی به پیاده‌سازی این توابع نیست (مگر اینکه برای حل مسئله به ساختمان داده یا صف خاصی نیاز داشته باشید که با آنچه در کلاس ارائه شده متفاوت باشد).

الف) فرض کنید یک اسب در یک خانه از شترنج قرار دارد. آیا مسیری وجود دارد که اسب در شترنج حرکت کرده (به صورت L) و هر خانه شترنج را یک و فقط یکبار ملاقات کند؟

ب) فرض کنید یک سرور وجود دارد که به کلاینت‌های مختلف سرویس می‌دهد. کلاینت‌ها ۴ اولویت مختلف دارند. می‌خواهیم سرور به احتمال ۰.۵۰٪ به کلاینتهای با اولویت ۱ پاسخ دهد، به احتمال ۰.۳۰٪ به کلاینت‌های با اولویت ۲، به احتمال ۰.۲۰٪ کلاینت‌های با اولویت ۳ و به احتمال ۰.۱۰٪ به کلاینت‌های با اولویت ۴. پس از انتخاب اولویت

ج) یک مسئله حساس بلادرنگ داریم که باید در کمترین زمان ممکن وجود یا عدم وجود یک عدد را به ما اعلام نماید.

۶. عناصری با کلیدهای زیر را به یک **max-heap** اضافه کردیم. درخت نهایی به چه صورت خواهد شد؟ سپس ۳ عنصر را از این درخت حذف کردیم. درخت پس از حذف به چه صورت خواهد شد؟ (قدم به قدم بکشید) (۲.۵ نمره)

۱۳, ۵, ۳, ۸, ۱۲, ۱۱, ۶, ۹, ۱۱, ۲۰

۷. می‌خواهیم یک ماتریس خلوت را به صورت لیست‌ها پیاده سازی نماییم. به طوریکه یک لیست پیوندی برای هر سطر ایجاد شود که هر کدام از عناصر آن به لیست پیوندی دیگر شامل عناصر سطر مربوطه در آرایه اشاره می‌کند. یک ساختمان داده برای اینکار پیشنهاد دهید (همراه کد) و الگوریتم دسترسی به عناصر ماتریس را بنویسید (۳ نمره).

۸. تابعی بنویسید که اشاره گر به عنصر اول به دو لیست پیوندی که در آن دو چند جمله‌ای خلوت ذخیره شده است را بگیرد و جمع آن دو چند جمله‌ای را محاسبه نماید (۲.۵ نمره)

۹. می خواهیم عبارت میانوندی زیر را به پسوندی تبدیل کنیم. ترتیب پر و خالی شدن پشته را در جدول زیر پر نمایید (۲ نمره)

$$a + (b - c) * d + e$$

Token	Stack[0]	Stack[1]	Stack[2]	Stack[3]	Stack[4]	Stack[5]	Top	Output
a								
+								
(
b								
-								
c								
)								
*								
d								
+								
e								

خلاصه مرتبه‌های زمانی

Algorithm	Best Time Complexity	Average Time Complexity	Worst Time Complexity	Worst Space Complexity
Linear Search	$O(1)$	$O(n)$	$O(n)$	$O(1)$
Binary Search	$O(1)$	$O(\log n)$	$O(\log n)$	$O(1)$
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Merge Sort	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(n)$
Quick Sort	$O(n\log n)$	$O(n\log n)$	$O(n^2)$	$O(\log n)$
Heap Sort	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(n)$
Bucket Sort	$O(n+k)$	$O(n+k)$	$O(n^2)$	$O(n)$
Radix Sort	$O(nk)$	$O(nk)$	$O(nk)$	$O(n+k)$
Tim Sort	$O(n)$	$O(n\log n)$	$O(n\log n)$	$O(n)$
Shell Sort	$O(n)$	$O((n\log(n))^2)$	$O((n\log(n))^2)$	$O(1)$

مجموعه های مجزا

مجموعه های مجزا (ساختمان داده)

فرض کنید مجموعه تاریخی ای از عناصر را در اختیار داریم. گاهی از موقع می خواهیم آن ها را به چند مجموعه بدون اشتراک افزای کنیم. داده ساختار مجموعه های مجزا، داده ساختاری است که این کار را انجام می دهد. یک الگوریتم جستجو-ادغام، الگوریتمی است که دو عمل زیر را روی این داده ساختار انجام می دهد:

جستجو: مشخص می کند که یک عنصر در کدام مجموعه قرار دارد. همچنین می تواند مشخص کند که آیا دو عنصر در یک مجموعه قرار دارند یا نه.

ادغام: دو مجموعه را با هم ادغام می کند و یک مجموعه جدید می سازد.

عملیات مهم دیگری هم که روی این داده ساختار انجام می شود، ایجاد مجموعه است که یک مجموعه جدید شامل یک عضو داده شده را ایجاد می کند که عموماً بسیار ساده است. با در اختیار داشتن این 3 عمل، می توان بسیاری از مسائل تقسیم بندی را حل کرد. برای این که این 3 عمل را با دقت تعریف کنیم، روشی برای مشخص کردن مجموعه ها لازم است. یک روش معمول برای این کار این است که برای هر مجموعه، یک عضو مشخص از آن را انتخاب کنیم و آن را نماینده این مجموعه بنامیم. در این صورت، تابع جستجو برای یک عضو، نماینده مجموعه ای که این عضو در آن قرار دارد را برمی گرداند و تابع ادغام نماینده دو مجموعه را به عنوان ورودی می گیرد.

یک روش ساده برای ساختن این داده ساختار به این شکل است که برای هر مجموعه، یک لیست پیوندی تشکیل دهیم و عنصر ابتدای هر لیست را به عنوان نماینده آن مجموعه انتخاب کینم. تابع ایجاد مجموعه، یک لیست جدید با یک عنصر را می سازد. تابع ادغام، لیست دو مجموعه را به هم پیوند می دهد، که زمان ثابتی می گیرد. مشکل این روش پیاده سازی در تابع جستجو است که به زمان ($\Omega(nm)$) بزرگ است. اما حالا تابع ادغام باید این اشاره گر به هر گره در لیست ها که به ابتدای آن لیست اشاره می کند، این مشکل را رفع کرد. اما حالا تابع ادغام باید این اشاره گر تمام گره های یک لیست که به لیست دیگری چسبانده می شود را به روز کند، پس این تابع به زمان ($\Omega(nm)$) احتیاج دارد. اگر طول هر لیست را نگه داریم، و در هر مرحله لیست کوچکتر را به لیست بزرگتر بچسبانیم، می توان زمان را بهبود داد. در این صورت اگر از m تابع جستجو، ادغام و ایجاد مجموعه روی n عضو اعمال شوند، زمان ($O(m + n \log n)$) مصرف می شود.

ادامه این بخش به زودی تکمیل می شود.