دانشکده علوم ریاضی و آمار

مدرس: دکتر مجتبی رفیعی　　　　نیمسال اول ۱۴۰۰-۱۴۰۱

ساختمان داده‌ها و الگوریتم‌ها

جلسه ۱۳ ساختمان داده‌ها و الگوریتم‌ها

نگارنده: فاطمه خورسند

۹ آبان ۱۴۰۰

فهرست مطالب

۱ مرور مفاهیم جانبی

فرض کنید $f(n) = logn!$ و $g(n) = n\log n$ باشد. نشان دهید که $f(n) = \theta\big(g(n)\big)$ می‌باشد.
طبق تعریف نماد مجانبی θ، می‌دانیم که

$$f(n) = \theta\big(g(n)\big) \iff f(n) = O(g(n)) \quad \text{and} \quad f(n) = \Omega(g(n))$$

نماد O را قبلا نشان داده‌ایم. طبق تعریفی که برای Ω گفتیم،

$$\exists\, c, n_0 \quad s.t. \quad \forall n \ \geq n_0 \quad , \quad c \times n\log n \leq \ \log n! \qquad (*)$$

راهنمایی:

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

$$\frac{1}{12n+1} \leq \alpha_n \leq \frac{1}{12n}$$

از دو طرف لگاریتم می‌گیریم و داریم:

$$\log n! = \frac{1}{2}\log(2\pi n) + n\log(\frac{n}{e}) + \alpha_n \log e$$

$$= \frac{1}{2}\log(2\pi) + \frac{1}{2}\log n + n\log n - n\log e + \alpha_n \log e \ \geq c \times n\log n$$

$\alpha_n \log e$ و $\frac{1}{2}\log(2\pi)$ مثبت و ثابت $(constant)$ هستند. پس داریم:

$$A = c + \frac{1}{2}\log n + n\log n - n\log e = O(n\log n)$$

یعنی:

$$\exists\, n'_0, d \quad s.t. \quad \forall n \ \geq n'_0 \qquad A \leq \ d\, n\log n$$

$$(*) \quad \rightarrow \quad c \times n\log n \leq \ d\, n \log n$$

$$d \ = \ c+1$$

$$n'_0 \ = \ n_0$$

۲ مرتب‌سازی حبابی

انواع مرتب‌سازی‌هایی که تاکنون مورد بررسی قرار دادیم، عبارتند از:

۱. مرتب‌سازی درجی $(Insertion\ sort)$،

۲. مرتب‌سازی سریع $(Quick\ sort)$،

۳. مرتب‌سازی حبابی $(Bubble\ sort)$،

مرتب‌سازی حبابی: با ذکر مثال این مرتب‌سازی را شرح می‌دهیم. آرایه $A = 4, 2, 7, 5, 6, 1$ را در نظر بگیرید. در هر مرحله سعی می‌شود، دو عنصر آخر مرتب شود و عنصر کوچکتر در سمت چپ قرارگیرد و به همین ترتیب عنصر اولیه از ابتدا، دوتا دوتا با بقیه عناصر مقایسه می‌شود. در مرحله اول عنصر مینیمم در آرایه پیدا و در اولین خانه قرار گرفت.

مرحله ۱:

- $4, 2, 7, 5, \underline{6, 1}$
- $4, 2, 7, \underline{5, 1}, 6$
- $4, 2, \underline{7, 1}, 5, 6$
- $4, \underline{2, 1}, 7, 5, 6$
- $\underline{4, 1}, 2, 7, 5, 6$
- $\underline{1}, 4, 2, 7, 5, 6$

مرحله ۲:

- $1, 4, 2, 7, \underline{5, 6}$

- $1, 4, 2, 7, 5, 6$
- $1, 4, 2, 5, 7, 6$
- $1, 4, 2, 5, 7, 6$
- $1, 2, 4, 5, 7, 6$

در آخر مرحله دوم دو عنصر مرتب داریم. همین ترتیب را تا آخرین مرحله انجام می‌دهیم.

مرحله ۵:

- $1, 2, 4, 5, 6, 7$
- $1, 2, 4, 5, 6, 7$

قبل از شروع مرحله ۵، تعداد ۴ عنصر مرتب شده است. از آخر آرایه دوتا دوتا مقایسه می‌شوند و مقدار مینیمم شیفت می‌خورد تا در خانه اول (قسمت مرتب نشده) قرار گیرد. در مرحله پنجم تمام آرایه‌ها مرتب شده‌اند. درواقع برای مرتب شدن آرایه‌ها به $(n-1)$ مرحله نیاز است.
نکته: در پایان مرحله i _ام تعداد i عنصر ابتدایی مرتب می‌شوند.

۳ الگوریتم مرتب‌سازی حبابی

حبابی

شبه کد مربوط به الگوریتم مرتب‌سازی درجی در ادامه آورده شده است.

Algorithm 1 Bubble-sort(A[1..n])

```
1: for i = 1 to n − 1 do
2:     for j = n downto i + 1 do
3:         if (A[j] < A[j − 1]) then
4:             swap(A[j], A[j − 1])
```

این الگوریتم، یک الگوریتم درجاست چرا که حافظه اضافی که برحسب اندازه ورودی باشد نیاز ندارد و روی خود آرایه A کار می‌کند.

یادآوری- اثبات درستی

۱: تعیین خصیصه ناوردایی،

۲: طی کردن گام‌ها:

(آ) گام آغازین،

(ب) گام نگهداری،

(ج) گام پایان.

$LI1$ و $LI2$ را داریم، که $LI1$ حلقه بیرونی for (خطوط ۱ تا ۴) است و $LI2$ حلقه درونی for (خطوط ۲ تا ۴) است.

حلقه درونی $LI2$: باید خصیصه ناوردایی را درست انتخاب کنیم. در مرحله j-ام عنصر مینیمم در زیرآرایه $A = [j..n]$ در خانه j-ام قرار می‌گیرد. در شروع حلقه $j = n$ است، یعنی اگر $A[n..n]$ را در نظر بگیریم چون یک خانه است، پس شرط اجرا نمی‌شود. چون عنصر مینیمم در خود آن است و مقایسه‌ای صورت نمی‌گیرد. در ادامه، j را یکی کم می‌کنیم و $n-1$ می‌شود. حال باید n را با $n-1$ مقایسه کنیم و عنصر مینیمم در خانه $n-1$ قرار گیرد. به همین ترتیب پیش می‌رویم تا به شرط پایان حلقه که $j = i$ است برسیم. نتیجه می‌گیریم از 1 تا n در آرایه، کوچکترین عنصر در خانه i قرار گرفته است.

حلقه بیرونی $LI1$: شبیه قبل، خصیصه ناوردایی و گام‌های ناوردایی را می‌بایست تعیین کنیم:
خصیصه ناوردایی: در پایان مرحله i _ ام، i عنصر ابتدایی آرایه A یعنی $A[1..i]$ مرتب می‌شوند.

- **گام آغاز:** $i = 1$ می‌باشد، پس $A[1..1]$ یک خانه دارد، پس مرتب است.
- **گام نگهداری:** در مرحله $i = k$ می‌دانیم از 1 تا $k - 1$ مرتب است و در پایان آن مرحله $A[1..k]$ مرتب است.
- **گام پایانی:** $i = n$ می‌باشد، پس $A[1..n]$ مرتب است.

دانشکده علوم ریاضی و آمار

مدرس: دکتر مجتبی رفیعی نیمسال اول ۱۴۰۰-۱۴۰۱

ساختمان داده‌ها و الگوریتم‌ها

جلسه ۱۶

نگارنده: زهرا درویشی

۱۶ آبان ۱۴۰۰

فهرست مطالب

۱ الگوریتم مرتب سازی ادغامی

شبه کد مربوط به الگوریتم مرتب سازی ادغامی که حاوی دو تابع Merge-Sort و Merge است، در ادامه آورده شده است.

Algorithm 1 Merge-Sort$(A[1\cdots n], p, r)$

if $(p < r)$ **then**
  $q \leftarrow \left\lfloor \frac{(p+r)}{2} \right\rfloor$
  Merge-Sort $(A[1\cdots n], p, q)$
  Merge-Sort $(A[1\cdots n], q+1, r)$
  Merge $(A[1\cdots n], p, q, r)$

Algorithm 2 Merge($A[1\cdots n], p, q, r$)

1: $\triangleright$ Assume that $A[p\cdots q]$ and $A[q+1\cdots r]$ are sorted.
2: $n_1 \leftarrow q - p + 1$
3: $n_2 \leftarrow r - q$
4: Let $L[1\cdots n_1+1]$ and $R[1\cdots n_2+1]$ be two empty arrays.
5: **for** $i = 1$ to n_1 **do**
6:     $L[i] \leftarrow A[p+i-1]$
7: **for** $i = 1$ to n_2 **do**
8:     $R[i] \leftarrow A[q+i]$
9: $L[n_1+1] \leftarrow \infty, \quad R[n_2+1] \leftarrow \infty$
10: $i \leftarrow 1, \quad j \leftarrow 1$
11: **for** $k = p$ to r **do**
12:     **if** $(L[i] \leq R[j])$ **then**
13:         $A[k] \leftarrow L[i]$
14:         $i \leftarrow i + 1$
15:     **else**
16:         $A[k] \leftarrow R[j]$
17:         $j \leftarrow j + 1$

۱.۱ اثبات درستی الگوریتم

اثبات الگوریتم، شامل دو قسمت است:

۱۱ تا ۱۷

- اثبات درستی الگوریتم Merge، توسط حلقه ناوردایی مربوط به خطهای 10 تا 16 انجام می شود. در این راستا، خصیصه ناوردایی را به این صورت تعریف می‌کنیم: زیر آرایه $A[p...k-1]$ شامل $k-p$ کوچکترین عنصر آرایه‌های $L[1...n_1+1], \; R[1...n_2+1]$ به صورت مرتب است و همچنین $L[i], \; R[j]$ کوچکترین عناصر آرایه‌های خودشان هستند که در A کپی نشده‌اند. به راحتی می‌توانید گام‌های ناوردایی حلقه را برای خصیصه ذکر شده دنبال کنید. از اینرو ما از ذکر دقیق آنها خودداری کرده‌ایم.
- برای کل الگوریتم نیز از اثبات استقرایی استفاده می‌کنیم.
  - **پایه استقرا:** آرایه تک عنصری، یک آرایه مرتب است.
  - **گذار استقرا:** هر دو زیر آرایه با اندازه $\frac{n}{2}$ مرتب است و الگوریتم Marge هم نشان دادیم صحیح است، پس آرایه با اندازه n مرتب است.

۲.۱ تحلیل پیچیدگی الگوریتم

طبق فرم کلی بیان شده برای الگوریتم‌های تقسیم و غلبه:

$$T(n) = aT(\tfrac{n}{b}) + D(n) + C(n)$$
$$T(1) = c$$

داریم:

$$a = 2, \quad b = 2, \quad D(n) = \mathcal{O}(1), \quad C(n) = \mathcal{O}(n)$$

بنابراین:

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

که $\mathcal{O}(1)$ جذب $\mathcal{O}(n)$ می‌شود و در نهایت داریم:

$$T(n) = 2T(\frac{n}{2}) + \mathcal{O}(n)$$

دانشکده علوم ریاضی و آمار

مدرس: دکتر مجتبی رفیعی — نیمسال اول ۱۴۰۰-۱۴۰۱

ساختمان داده‌ها و الگوریتم‌ها

جلسه ۱۷: حل روابط بازگشتی

نگارنده: مریم نظری‌راد

۱۷ آبان ۱۴۰۰

فهرست مطالب

۱ حل روابط بازگشتی

حل یک رابطه بازگشتی بدین معناست که به جای نوشتن تابع برحسب جملات قبلی، یک مقدار مشخص برای آن تابع برحسب پارامتر ورودی آن به دست آوریم. با اینحال، یک روش کلی برای حل تمام توابع بازگشتی وجود ندارد و متناسب با فرم تابع بازگشتی تاکنون راه‌حل هایی ارائه شده است. برخی از این روش‌ها عبارتند از:

۱. روش تلسکوپی،

۲. روش معادله مشخصه،

۳. روش حدس و استقرا،

۴. روش جایگذاری،

۵. روش درخت بازگشت.

تمرکز ما در این درس، بر روی روش‌های زیر است:

- دس و استقرا،
- جایگذاری،
- درخت بازگشت،
- قضیه اساسی [1].

۱.۱ روش حدس و استقرا

یادآوری استقراء قوی: گاهی اوقات برای اثبات حکم استقرا به ازای n، لازم است که حکم استقرا را به ازای هر عدد صحیح کوچکتر از n و بزرگتر یا مساوی m (که m پایه استقراست) صحیح فرض کنیم. به چنین مفهومی استقرای قوی می‌گوییم که به طور دقیق تعریف آن در ادامه آورده شده است.

تعریف ۱ (استقرا قوی) *فرض کنید $p(n)$ حکمی در مورد اعداد طبیعی باشد. برای اثبات صحت حکم برای مقادیر طبیعی n، یک اثبات استقراء قوی صحت گزاره‌های زیر را تایید می‌کند.*

۱. *پایه استقراء: p(m) درست است.(که m نمونه کوچک ماست)*

۲. *گام استقراء: به ازای هر عدد طبیعی $k \geq m$، اگر p(k) درست باشد آن‌گاه p(k+1) نیز درست است.*

مثال روش حدس و استقراء: تابع زیر را در نظر بگیرید

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

حدس ما می‌تواند یک کران بالا یا یک کران پایین برای رابطه بازگشتی فوق باشد.

- **حدس ۱:** پیدا کردن یک کران بالا برای $T(n)$:

  گام ۱: حدس:

  $$\forall\, n \geq 2 \quad T(n) \leq 4n$$

  گام ۲: بررسی و طی کردن گام‌های استقرا:

  پایه استقرا:

  $$n = 2 \longrightarrow T(2) = 1 \leq 4 \times 2 = 8$$

  گام استقرا: فرض می‌کنیم:

  $$\forall 2 < k < n \quad T(k) \leq 4k$$

  باید بررسی کنیم اگر $k = n$ آنگاه $T(n) \leq 4n$؟

  می‌دانیم $\frac{n}{2} < n$ بنابراین طبق فرض استقراء داریم

  $$T(\frac{n}{2}) \leq (\frac{4n}{2})$$

  در نتیجه

  $$T(n) = 2T(\frac{n}{2}) + n \leq 2 \times 4 \times \frac{n}{2} + n = 5n \longrightarrow T(n) \leq 5n \leq 4n$$

  بنابراین حدس ۱ اشتباه است.

[1] Master theorem

- **حدس ۲:** پیدا کردن یک کران بالا برای $T(n)$:

گام ۱: حدس:

$$\forall n \geq 2 \quad T(n) \leq 5n$$

گام ۲: بررسی و طی کردن گام‌های استقرا:

پایه استقرا:

$$n = 2 \longrightarrow T(2) = 1 \leq 5 \times 2 = 10$$

گام استقرا: فرض می‌کنیم

$$\forall 2 < k < n \quad T(k) \leq 5k$$

باید بررسی کنیم اگر $k = n$ آنگاه $T(n) \leq 5n$؟

می‌دانیم $\frac{n}{2} < n$ بنابراین طبق فرض استقرا داریم:

$$T(\frac{n}{2}) \leq (\frac{5n}{2})$$

در نتیجه

$$T(n) = 2T(\frac{n}{2}) + n \leq 2 \times 5 \times \frac{n}{2} + n = 6n \longrightarrow T(n) \leq 6n \leq 5n$$

بنابراین حدس ۲ اشتباه است.

جمع‌بندی

جمع بندی از حدس ۱ و ۲: $T(n)$ خطی نیست و یک تابع غیرخطی است. در حدس ۱و۲ می‌توانستیم به جای عدد۴ و ۵، عدد c بگذاریم و به طور کلی نتیجه‌گیری کنیم که خطی نیست.

- **حدس ۳:** پیدا کردن یک کران بالا برای $T(n)$:

گام۱: حدس:

$$\forall n \geq 2 \quad T(n) \leq cn \quad \boxed{c < 0}$$

با توجه به اینکه در ادامه
c
را یک مقدار مثبت در نظر گرفته است
پس در ابتدا نمیتوان گفت
c<0

گام۲: بررسی و طی کردن گام‌های استقرا:

پایه استقرا:

$$n = 2 \longrightarrow T(2) = 1 \leq c \times 2 = 2c$$

گام استقرا: فرض می‌کنیم

$$\forall\, 2 < k < n \quad T(k) \leq ck$$

باید بررسی کنیم اگر $k = n$ آنگاه $T(n) \leq cn$ ؟

می‌دانیم $\frac{n}{2} < n$ بنابراین طبق فرض استقرا داریم

$$T(\frac{n}{2}) \leq (\frac{cn}{2})$$

در نتیجه

$$T(n) = 2T(\frac{n}{2}) + n \leq 2 \times c \times \frac{n}{2} + n = (c+1)n \longrightarrow T(n) \leq (c+1)n \leq cn$$

به ازای هر $c > 0$ که انتخاب این رابطه برقرار نیست. پس این حدس اشتباه است و تابع T(n) خطی نیست.

نکته

اگر حدس درست بود، باید c را به نحوی پیدا می‌کردیم که هم پایه و هم گام استقرا رابطه‌های نوشته شده برایشان صحیح باشند.

- **حدس ۴:** پیدا کردن یک کران بالا برای $T(n)$:

گام۱: حدس:

$$\forall n \geq 2 \quad T(n) \leq cn^2 \quad \boxed{c < 0}$$

با توجه به اینکه در ادامه
c
را برابر 2 در نظر گرفتیم
نمیتوان در ابتدا
c<0
باشد

گام ۲: بررسی و طی کردن گام‌های استقرا:

پایه استقرا:

$$n = 2 \longrightarrow T(2) = 1 \leq c \times 2^2 = 4c$$

گام استقرا: فرض می‌کنیم

$$\forall 2 < k < n \quad T(k) \leq ck^2$$

باید بررسی کنیم اگر $n = k$ آنگاه $T(n) \leq cn^2$ ؟

می‌دانیم $\frac{n}{2} < n$ بنابراین طبق فرض استقرا داریم:

$$T(\frac{n}{2}) \leq c(\frac{n}{2})^2$$

در نتیجه

$$T(n) = 2T(\frac{n}{2}) + n \leq 2 \times c \times (\frac{n}{2})^2 + n = \frac{cn^2}{2} + n \longrightarrow T(n) \leq \frac{cn^2}{2} + n \leq cn^2$$

اگر فرض کنیم $c = 2$ باشد، آنگاه داریم $n^2 + n \leq 2n^2$، و همچنین برای پایه استقراء هم برقرار است چرا که $T(2) = 1 \leq 2 \times 2^2 = 8$. بنابراین نشان دادیم وقتی $c = 2$ باشد،

$$T(n) \leq cn^2 \longrightarrow T(n) = O(n^2)$$

نکته

برای پیدا کردن حدس خوب (پیدا کردن کوچکترین حد بالا)، تغییر ثابت c مطلوب نیست و در معرفی نمادهای مجانبی علت را متوجه شدیم. پس بهتر است برای حدس خوب روی n (اندازه ورودی) کار کنیم.

نکته

در حدهای قبل دیدیم که حدس cn اشتباه بود و حدس cn^2 درست بود اما ممکن است بهینه نباشد. در حدس بعدی سعی داریم یک پیچیدگی زمانی بین این دو را بررسی کنیم. می دانیم:

$$cn \leq cn \log n \leq cn^2$$

.

با توجه به اینه در ادامه
c
را برابر 1 در نظر گرفتیم
در ابتدا نمیتوانیم
c <0
فرض کنیم.

- **حدس ۵:** پیدا کردن یک کران بالا برای $T(n)$:

گام ۱: حدس:

$$\forall n \geq 2 \quad T(n) \leq cn \log n \quad \boxed{c < 0}$$

گام ۲: بررسی و طی کردن گام‌های استقرا:

پایه استقرا:

$$n = 2 \longrightarrow T(2) = 1 \leq 2c \log 2 = 2c$$

گام استقرا: فرض می‌کنیم:

$$\forall 2 < k < n \quad T(k) \leq ck \log k$$

باید بررسی کنیم اگر $k = n$ آنگاه $T(n) \leq cn \log n$ ؟

می‌دانیم $\frac{n}{2} < n$، بنابراین طبق فرض استقرا داریم:

$$T(\frac{n}{2}) \leq c\frac{n}{2} \log(\frac{n}{2})$$

در نتیجه

$$T(n) = 2T(\frac{n}{2}) + n \leq 2 \times c \times \frac{n}{2} \times \log(\frac{n}{2}) + n = cn(\log n - \log 2) + n$$

$$T(n) \leq cn \log n - cn + n \leq cn \log n$$

اگر فرض کنیم $c = 1$ آنگاه داری

$$n \log n \leq n \log n$$

و همچنین برای پایه استقرا هم برقرار است چرا که

$$T(2) = 1 \leq 2 \log 2 = 2$$

،

بنابراین نشان دادیم وقتی $c = 1$ باشد، آنگاه

$$T(n) \leq cn \log n \longrightarrow T(n) = O(n \log n)$$

تا این‌جا حدس‌مان برای پیدا کردن یک کران بالا برای $T(n)$ بود، در ادامه می‌خواهیم یک کران پایین از آن حدس بزنیم.

- **حدس ۶:** پیدا کردن یک کران پایین برای $T(n)$:

گام ۱: حدس:

$$\forall n \geq 2 \quad T(n) \geq cn \log n$$

گام ۲: بررسی و طی کردن گام‌های استقرا:

پایه استقرا:

$$n = 2 \longrightarrow T(2) = 1 \geq 2c \log 2 = 2c$$

گام استقرا: فرض می‌کنیم:

$$\forall 2 < k < n \quad T(k) \geq ck \log k$$

باید بررسی کنیم اگر $k = n$ باشد، آنگاه $T(n) \geq cn\log n$ ؟
می‌دانیم $n > \frac{n}{2}$ بنابراین طبق فرض استقرا داریم:

$$T(\frac{n}{2}) \geq c\frac{n}{2}\log(\frac{n}{2})$$

در نتیجه

$$T(n) = 2T(\frac{n}{2}) + n \geq 2 \times c \times \frac{n}{2} \times \log(\frac{n}{2}) + n = cn(\log n - \log 2) + n$$

$$T(n) \geq cn\log n - cn + n \geq cn\log n$$

اگر فرض کنیم $c = 1$ باشد، داریم:

$$n\log n \geq n\log n$$

با اینحال، پایه استقرا برقرار نیست و $T(2) = 1 \geq 2\log 2 = 2$.
اگر فرض کنیم $c = \frac{1}{2}$ باشد، داریم:

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

و همچنین پایه نیز بر قرار است چرا که $T(2) = 1 \geq 1$.
بنابراین نشان دادیم وقتی $c = \frac{1}{2}$ باشد،

$$T(n) \geq cn\log n \longrightarrow T(n) = \Omega(n\log n)$$

.

جمع‌بندی

جمع بندی از حدس ۵ و ۶: در این حدس‌ها نشان دادیم که
$T(n) \leq cn\log n \longrightarrow T(n) = O(n\log n)$
$T(n) \geq cn\log n \longrightarrow T(n) = \Omega(n\log n)$

و از آنجایی که
$f(n) = O(g(n)) \quad and \quad f(n) = \Omega(g(n)) \Longleftrightarrow f(n) = \theta(g(n))$
داریم:
$T(n) = \theta(n\log n)$.

نکته

همان‌طور که دیدیم لزومی ندارد که c ای که در حدس ۵ و حدس ۶ استفاده کردیم یکسان باشد.(طبق تعریف Ω و $\mathcal{O}$ واضح است).

دانشکده علوم ریاضی و آمار

مدرس: دکتر مجتبی رفیعی	نیمسال اول ۱۴۰۰-۱۴۰۱

ساختمان داده‌ها و الگوریتم‌ها

جلسه ۱۸: ساختمان داده و الگوریتم‌ها

نگارنده: عاطفه قوقه‌ای

۸ آبان ۱۴۰۰

فهرست مطالب

۱ حل روابط بازگشتی-روش جایگذاری

مثال برای روش جایگذاری: تابع بازگشتی زیر را که در یک فرم کلی داده شده است در نظر بگیرید:

$$T(n) = aT(\frac{n}{b}) + f(n)$$

گام اول جایگذاری:

؟

$$\begin{aligned} T(n) &= aT(\frac{n}{b}) + f(n) \\ &= a(aT(\frac{n}{b^2}) + \boxed{F}(\frac{n}{b})) + f(n) \\ &= a^2T(\frac{n}{b^2}) + af(\frac{n}{b}) + f(n) \end{aligned}$$

گام دوم جایگذاری :

$$T(n) = a^2T(\frac{n}{b^2}) + af(\frac{n}{b})) + f(n)$$

$$= a^2(aT(\frac{n}{b^3}) + \overset{f}{\boxed{F}}(\frac{n}{b^2})) + a\overset{f}{\boxed{F}}(\frac{n}{b}) + \overset{f}{\boxed{F}}(n)$$

$$= a^3T(\frac{n}{b^3}) + a^2\underset{f}{\boxed{F}}(\frac{n}{b^2})) + a\underset{f}{\boxed{F}}(\frac{n}{b}) + \underset{f}{\boxed{F}}(n)$$

گام K-ام جایگذاری

$$T(n) = a^kT(\frac{n}{b^k}) + \sum_{i=0}^{k-1} a^i\overset{f}{\boxed{F}}(\frac{n}{b^i})$$

حال سوالی که پیش می‌آید این است که تا کجا باید k را پیش ببریم؟ تا جایی که $T(\frac{n}{b^k})$ به حالت پایه که برابر یک است برسد.

$$\frac{n}{b^k} = 1 \qquad \longmapsto \qquad n = b^k \qquad \longmapsto \qquad \log_b n = k$$

پس رابطه کلی زیر حاصل می‌شود:

$$T(n) = a^{\log_b n}T(\frac{n}{b^{\log_b n}}) + \sum_{i=0}^{(\log_b n)-1} a^i f(\frac{n}{b^i})$$

از آن جایی که $n^{\log_b a} = a^{\log_b n}$ است می‌توان عبارت بالا را به صورت زیربازنویسی کرد:

$$T(n) = n^{\log_b a}T(1) + \sum_{i=0}^{(\log_b n)-1} a^i f(\frac{n}{b^i})$$

رابطه‌ی بازگشتی ما دیگر براساس جملات قبلی نیست بلکه براساس اندازه ورودی آن است. حال در ادامه قصد داریم دو حالت خاص از فرم کلی رابطه بالا را در نظر گرفته و حل کنیم:
ابتدا فرض کنید که $f(n) = C$ باشد بنابراین داریم:

$$T(n) = n^{\log_b a}T(1) + \sum_{i=0}^{(\log_b n)-1} Ca^i$$

$$= \Theta(n^{\log_b a}) + \Theta(n^{\log_b a}) = \Theta(n^{\log_b a})$$

$$\underbrace{n^{\log_b a}T(1)} \qquad \longmapsto \qquad \Theta(n^{\log_b a})$$

$$\underbrace{\sum_{i=0}^{(\log_b n)-1} Ca^i} \qquad = C\sum_{i=0}^{(\log_b n)-1} a^i = C\frac{a^{(\log_b n)-1+1}-1}{a-1}$$

$$= C\frac{n^{\log_b a}-1}{a-1} = \Theta(n^{\log_b a})$$

نکته: برای هر عدد حقیقی چون $a \neq 1$ باشد داریم:

$$\sum_{i=0}^{u} a^i = \frac{a^{u+1}}{a-1}$$

فرض کنید $f(n) = n$ باشد بنابراین داریم:

$$T(n) = aT(\frac{n}{b}) + f(n)$$

$$T(n) = n^{\log_b a} T(1) + \sum_{i=0}^{(\log_b n)-1} a^i f(\frac{n}{b^i})$$

$$\sum_{i=0}^{(\log_b^n)-1} a^i * \frac{n}{b^i} = n \sum_{i=0}^{(\log_b n)-1} (\frac{a}{b})^i = n * \frac{(\frac{a}{b})^{\log_b n} - 1}{\frac{a}{b} - 1}$$

$$(\frac{a}{b})^{\log_b n} = n^{\log_b \frac{a}{b}} = n^(\log_b a - \log_b b) = n^{\log_b a - 1} = O(n^{\log_b a})$$

نکته: اگر روابط بازگشتی پیچیده‌تر شود مثلا تابع بازگشتی براساس دو تابع کوچکتر تعریف شود:

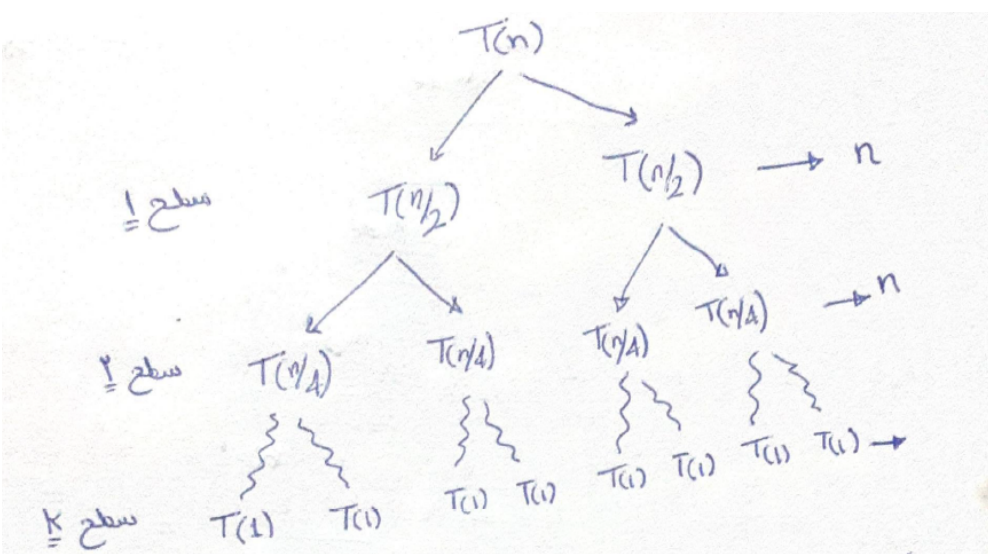
$$T(n) = 2T(\frac{n}{2}) + 3T(\frac{2n}{5}) + n$$

در این حالت روش جایگذاری آزاردهنده بوده و ممکن است نتوان از این روش بهره گرفت.

۲ حل روابط بازگشتی-روش درخت بازگشت

می‌توان برای حل روابط بازگشتی درخت آن را ترسیم کرد که به آن درخت بازگشت گفته می‌شود.
مثال۱.

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



حال سوال این است که تا کجا پیش برویم؟

$$\frac{n}{2^k} = 1 \quad \longmapsto \quad n = 2^k \quad \longmapsto \quad \log_2 n = k$$

تعداد نودها در سطح k-ام:

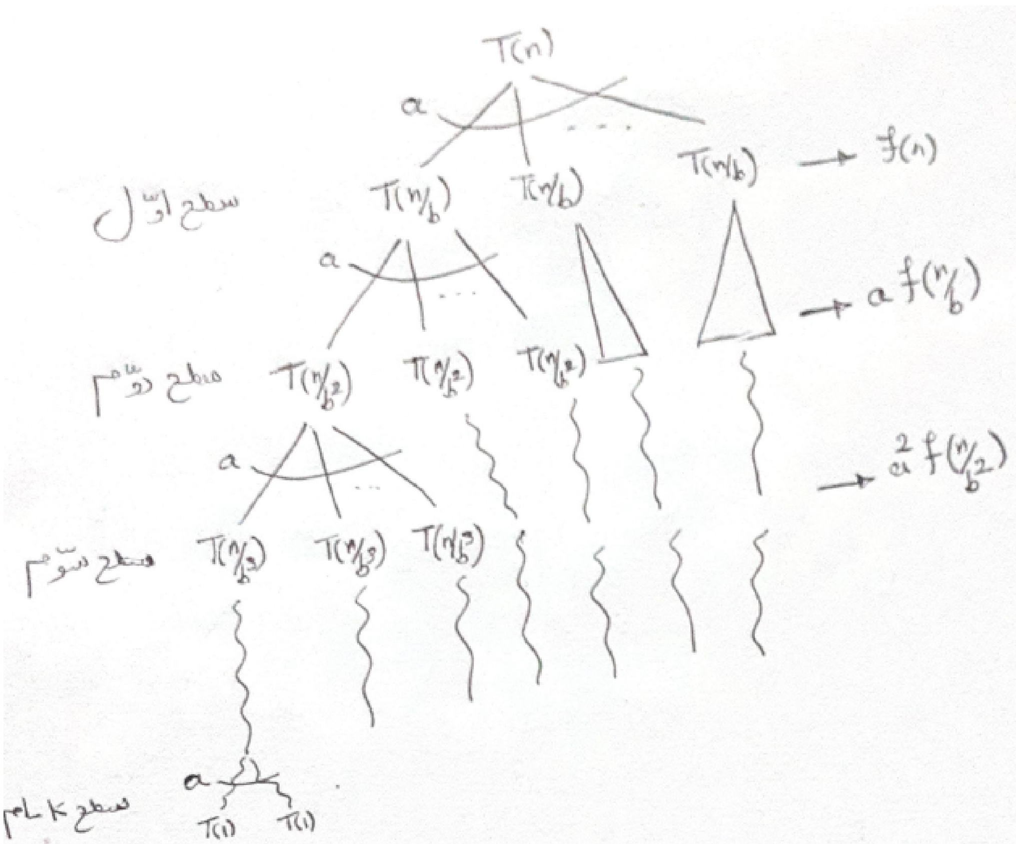
$$2^k = 2^{\log_2 n} = n$$

$$x = nT(1) + \frac{n}{2} * 2$$

$$T(n) = nT(1)+n+\sum_{i=1}^{k-1} n = nT(1)+n+n(k-1) = nT(1)+n+n(\log_2 n-1) = nT(1)+n\log_2 n = \Theta(n\log_b n)$$

مثال۲.

$$T(n) = aT(\frac{n}{b}) + f(n)$$



تا کجا پیش می‌رویم؟

$$\frac{n}{b^k} = 1 \qquad \longmapsto n = b^k \qquad \longmapsto \log_b n = k$$

تعداد نودها در سطح kام:

$$a^k = a^{\log_b n} = n^{\log_b a}$$

تابع درF مرحله k:

$$f(\frac{n}{b^k}) = f(\frac{n}{b^{\log_b n}}) + f(\frac{n}{n}) = f(1)$$

تابع f در مرحله K ام

$$T(n) = n^{\log_b a}T(1) + a^{(\log_b n)-1}f(1) + \sum_{i=1}^{k-1} a^{i-1} f(\frac{n}{b^{i-1}})$$

دانشکده علوم ریاضی و آمار

مدرس: **دکتر مجتبی رفیعی** نیمسال اول ۱۴۰۰-۱۴۰۱

ساختمان داده‌ها و الگوریتم‌ها

جلسه ۱۹

نگارنده: شقایق اسماعیلیان

۹ آبان ۱۴۰۰

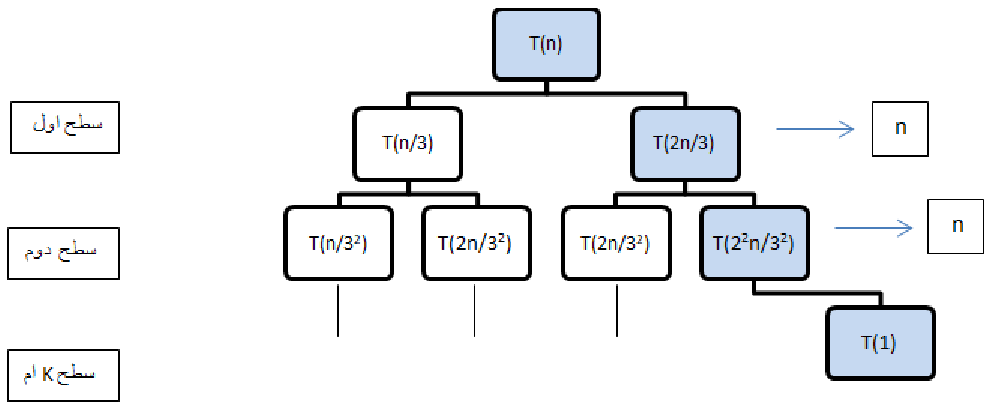
فهرست مطالب

۱ استفاده از دیدگاه مجانبی

همواره نیاز نیست رابطه بازگشتی را به صورت دقیق حل کنیم، بلکه می‌توانیم در مواردی یک حد بالای معقول برای آن پیدا کنیم. به عنوان مثال، رابطه زیر را در نظر بگیرید:

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

درختی که نامتوازن است و برخی مسیرها سریعتر به برگ می‌رسند.



تا کجا باید پیش بریم (مسیر طولانی تر):

$$(\frac{2}{3})^k n = 1 \Rightarrow \frac{n}{(3/2)^k} = 1 \Rightarrow n = (\frac{3}{2})^k$$

تعداد نودها در سطح k:

$$2^k = 2^{log^{2}_{\frac{3}{2}}} = n^{log^{2}_{\frac{3}{2}}}$$

حد بالا: $T(n) = O(nlog^n) = O(nlog^n_{\frac{3}{2}})$

حد پائین هم می‌توان گرفت: شاخه با رشد کمتر:

$$T(n) = \Omega(nlog^n) = \Omega(nlog^n_3)$$

نکته: نشان دهید رابطه $log^n_b = O(log^n)$ برقرار است.

$$\exists c, n > 0\ \forall n \geq n_0 0 \leq log^n_b \leq clog^n$$

می‌دانیم که $log^a_b = \frac{log^a_c}{log^b_c}$، بنابراین برای $0 \leq log^n_b \leq clog^n$ داریم:

$$0 \leq \frac{log^n}{log^b} \leq clog^n$$

و کافی است که $c = log^b$ فرض کنیم.

نکته: درخت بازگشت، روش خوبی برای اثبات نیست، بلکه روش خوبی برای حدس زدن است و برای اثبات دقیق می‌بایست با استفاده از روش حدس و استقرا، حل رابطه بازگشتی را انجام دهیم.

۲ روش ابتکاری

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

تغییر متغیر: $2^m = n$ بنابراین $m = log^n_2$ و داریم:

$$T(2^m) = 2T\boxed{(\frac{2^m}{2})} + m$$

۲

$(۲^{\frac{m}{۲}})$

اگر $T(2^m) = S(m)$ خواهیم داشت: $S(m) = 2S(\frac{m}{2}) + m$ که فرم آشنایی است و قبلا آن را حل کردیم (روش حدس و استقرا و روش جایگزینی)،

$$S(m) = \theta(mlog^m)$$

از آنجائیکه $S(m) = T(2^m)$ داریم:

$$T(2^m) = \theta(mlog^m)$$

و از آنجائیکه $m = log_2^n$ بود، داریم:

$$T(2^m) = T(n) = \theta(log^n log(log^n))$$

۳ قضیه اصلی (Master Theorem)

در معرفی رویکرد تقسیم و غلبه دیدیم که پیچیدگی زمانی چنین الگوریتم‌هایی در فرم زیر قابل بیان است.

$$T(n) = aT(n/b) + D(n) + C(n)$$

واضح است که برای حل چنین رابطه بازگشتی، می‌توان فرم کلی زیر را برای عبارت بالا بازنویسی کرد:

$$T(n) = aT(n/b) + f(n) \qquad (*)$$

در این بخش با معرفی قضیه اصلی سعی داریم تا به صورت کارایی برای حالت های خاصی از $f(n)$، رابطه بازگشتی $(*)$ را حل کنیم.

۱.۳ بحث روی مقادیر a و b در رابطه بازگشتی $(*)$

* **فرض کنید** $0 < b < 1$ **باشد:** اگر چنین باشد، هر بار برای حل $T(n)$ سراغ یک n بزرگتر می‌رویم. $T(n) = aT(m = n/b) + f(n)$ جایی که $m > n$ است. در نتیجه، عملا روال بازگشتی هیچ‌وقت تمام نمی‌شود، از اینرو $0 < b < 1$ بی معنا است.

* **فرض کنید** $b = 1$ **باشد:** اگر چنین باشد، رابطه بازگشتی $T(n)$ به صورت بدیهی قابل محاسبه است:

$$T(n) = aT(n) + f(n)$$

$$\rightarrow T(n) - aT(n) = f(n) \rightarrow (1-a)T(n) = f(n) \rightarrow T(n) = f(n)/1-a$$

نکته: از آنجاییکه $T(n)$ باید مثبت باشد و $f(n)$ یک تابع صعودی است پس $0 \leq a < 1$ باید باشد.

* **فرض کنید** $0 < a < 1$ **باشد:** اگر چنین باشد، با بهره گیری از حل رابطه بازگشتی با استفاده از روش جایگذاری که قبلا محاسبه کردیم، داریم:

$$T(n) = aT(n/b) + f(n) = a^{log_b^n}.T(1) + \sum_{i=0}^{(log_b^a)-1} a^i f(n/b^i)$$

$$A^* = a^{log_b^n}.T(1)$$

از آنجاییکه $0 < a < 1$ مد نظر است مقدار $A^* < T(1)$ است و بنابراین داریم:

$$T(n) \leq T(1) + \sum_{i=0}^{(log_b^a)-1} a^i f(n/b^i)$$

$$B^* = T(1) + \sum_{i=0}^{(log_b^a)-1} a^i f(n/b^i)$$

همچنین از آنجاییکه در تحلیل پیچیدگی الگوریتم‌ها، به طور معمول $f(n)$ یک تابع صعودی است، داریم:

$$T(n) \le B^* \le T(1) + f(n)\sum_{i=0}^{(log_b^a)-1} a^i \le T(1) + f(n) * 1/1 - a = O(f(n))$$

جمع بندی: با توجه به بررسی های انجام شده در بالا، می‌توان قضیه اساسی را برای مقادیر $a \ge 1$ و $b > 1$ به صورت خوش فرمی که در ادامه آمده است، تعریف کرد.
قضیه: اگر رابطه بازگشتی به فرم زیر داشته باشیم:

$$T(n) = aT(n/b) + f(n), where, b > 1, a \ge 1$$

آنگاه:

$(n^c log^k n)$ $(n^c log^{k+1} n)$

- **حالت ۱:** اگر $f(n) = O(n^{c-\mathcal{E}})$ باشد، جائیکه $\mathcal{E} > 0$ آنگاه $T(n) = \theta(n^c)$.
- **حالت ۲:** اگر $f(n) = \boxed{\theta(n^c log_n^c)}$ باشد، جائیکه k یک عدد صحیح مثبت باشد، آنگاه $T(n) = \boxed{\theta(n^c log_n^{k+1})}$.
- **حالت ۳:** اگر $f(n) = \Omega(n^{c+\mathcal{E}})$ باشد، جائیکه $\mathcal{E} > 0$ است و همچنین $af(n/b) \le hf(n)$ باشد برای یک $0 < h < 1$ و هر n به اندازه کافی بزرگ، آنگاه $T(n) = \theta(f(n))$.

در هر سه حالت ذکر شده در بالا $c = log_b^a$ می‌باشد (با توجه به روش های قبلی ذکر شده برای حل روابط بازگشتی، علت این مقدار برای c واضح است).
توجه: برای مشخص کردن حالت های ۱،۲،۳، به نحوی بحث روی n^c است:

- یک $\mathcal{E}$ کمتر،
- یک log_n^k ضربدر n^c (تقریبا در مرتبه n^c)،
- یک $\mathcal{E}$ بیشتر.

سوال: آیا حالت‌های بیان شده در قضیه بالا، همه توابع $f(n)$ صعودی را پوشش می‌دهد؟ پاسخ منفی است، در ادامه با ذکر مثال هایی به این موضوع می‌پردازیم.

دانشکده علوم ریاضی و آمار

مدرس: دکتر مجتبی رفیعی نیمسال اول ۱۴۰۰-۱۴۰۱

ساختمان داده‌ها و الگوریتم‌ها

جلسه ۲۳: ساختمان داده و الگوریتم‌ها

نگارنده: فاطمه علیرضایی

۱۹ آبان ۱۴۰۰

فهرست مطالب

۱ عملیات روی لیست‌های پیوندی

کلیه عملیاتی که در این بخش معرفی می‌شوند بر پایه لیست پیوندی دوطرفه تشریح شده است، با این حال به‌راحتی قابل تبدیل به دیگر نوع‌ها نیز می‌باشد.

- List-Search(L،k): یک پرسمان بازیابی است که برای پیدا کردن اولین عنصری که مقدار کلید آن برابر k است مورد استفاده قرار می‌گیرد. اگر عنصر در لیست موجود باشد، آن عنصر برگردانده می‌شود، در غیر اینصورت مقدار Null خروجی خواهد بود.

Algorithm 1 List-Search(L, k)

```
x = L.head
while x ≠ Null and x.key ≠ k do
    x = x.next
Return x
```

پیچیدگی زمانی الگوریتم فوق در بدترین حالت O(n) است،جایی که n تعداد نودهای لیست پیوندی L است.

- List–Insert(L،x): یک پرسمان بروزرسانی است که عنصر x را به ابتدای لیست L اضافه می‌کند.

Algorithm 2 List-Insert(L, x)

1: $x.next = L.head$
2: **if** $(L.head \neq$ Null$)$ **then**
3: $\quad L.head.prev = x$
4: $L.head = x$
5: $x.prev =$ Null

حذف

پیچیدگی زمانی الگوریتم فوق O(۱) است.

- List–Delete(L،x): یک پرسمان بروزرسانی است که عنصر x را از لیست L ~~اضافه~~ می‌کند.

Algorithm 3 List-Delete(L, x)

1: **if** $(x.prev \neq$Null$)$ **then**
2: $\quad x.prev.next = x.next$
3: **else**
4: $\quad L.head = x.next$
5: **if** $(x.next \neq$Null$)$ **then**
6: $\quad x.next.prev = x.\cancel{next}$ preve

7. else
8. L.tail = x.preve
9. x.preve.Next = Null

خطوط 7 تا 9 را ننوشته اند

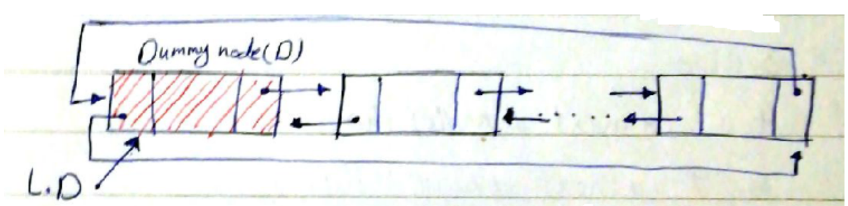
پیچیدگی زمانی الگوریتم فوق O(۱) است. نوشتن خطوط ۷ تا ۹ اجباری نیست و از لحاظ منطقی در خط ۲ بررسی شده است.

نکته: دقت کنید در حذف، چون عنصر x مدنظر بود پیچیدگی زمانی $O(1)$ شد. با این حال اگر مقدار کلید k مدنظر بود، می‌بایست اول آن را پیدا کنیم و سپس تغییرات را اعمال کنیم که در نهایت پیچیدگی زمانی آن $O(n)$ است. مطابق زیر:

Algorithm 4 List-Delete(L, k)

1: $x =$ List-Search(L, k)
2: List-Delete(L, x)

اطلاعات تکمیلی: اگر یک نود تصنعی (Dummy) در ابتدای لیست پیوندی دو طرفه اضافه کنیم و در عین حال آن را به صورت دایره‌ای نیز در نظر بگیریم، شبه کد مربوط به عملیات حذف ساده‌تر می‌شود.



شبه کد مربوط به بررسی تهی بودن لیست با مشخصات بالا:

$$L.D.next = L.D, \quad L.D.Prev = L.D$$

نکته: همیشه ایجاد نود ساختگی $(Dummy)$ممکن است سودمند نباشد و مثال بالا سبب ساده‌تر شدن الگوریتممان می‌شود. مثلا وقتی تعداد لیست‌های کوچک زیادی داریم، در نظر گرفتن نود ساختگی سبب اتلاف حافظه می‌شود.

- حذف نود x از لیست L:

Algorithm 5 DList-Delete(L, x)

```
x.prev.next = x.next
x.next.prev = x.prev
```

- جستجوی نود با کلید k در لیست L:

Algorithm 6 DList-Search(L, k)

```
x = L.D.next
while x ≠ L.D and x.key ≠ k do
    x = x.next
Return x
```

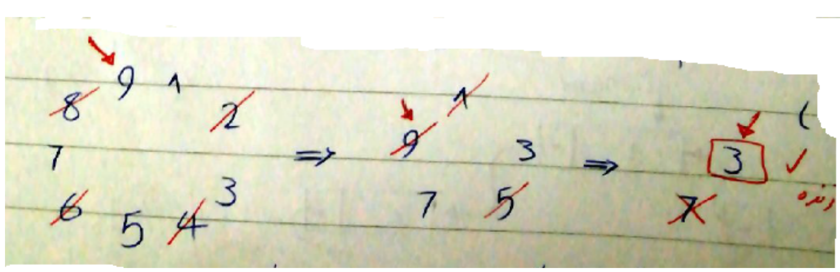
- درج نود x در ابتدای لیست L:

Algorithm 7 DList-Insert(L, x)

```
x.next = L.D.next
L.D.next.prev = x
L.D.next = x
x.prev = L.D
```

۲ کاربرد لیست‌ها - مساله جوزفوس

فرض کنید n نفر به صورت دایره‌وار ایستاده و منتظر اعدام هستند. مساله جوزفوس k (برای اعداد صحیح k >۱) به این طریق اعمال می‌کند که از ~~نظر~~ اول $k-1$ نفر رد کرده و نفر k را اعدام می‌کند و همین ترتیب تا زنده ماندن تنها 1 نفر ~~کار~~را ادامه می‌دهد. شکل زیر را به عنوان یک مثال برای این مساله در نظر بگیرید.
ا
نفر



mod n

برای مثال بالا، در نهایت نفری که در جایگاه ۳ قرار دارد زنده می‌ماند. به طور کلی، برای k و n فرمول بازگشتی زیر وجود دارد:

$$f(x,k) = (((f(n-1),k) \cancel{+k}) + k - 1)\boxed{mo\ dn}, \quad f(1,k) = 1$$

به طور ساده‌تر، برای $k = 2$ رابطه بازگشتی زیر ارائه شده است:

$$\begin{aligned} f(2n) &= 2f(n) + 1 && n \textit{ is even.} \\ \boxed{f(2+n+1)} &= 2f(n) - 1 && n \textit{ is odd.} \\ f(1) &= 1 && Base \end{aligned}$$

F(2n+1)

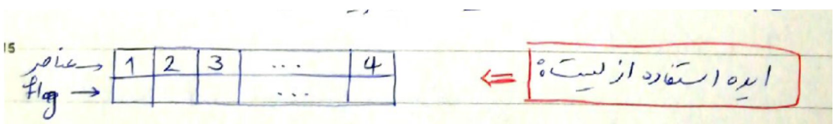
یک راه‌حل ساده برای $k = 2$:

- نوشتن رشته باینری معادل،
- شیفت دورانی به سمت چپ.

حل مثال بالا با روش مذکور به صورت زیر است:

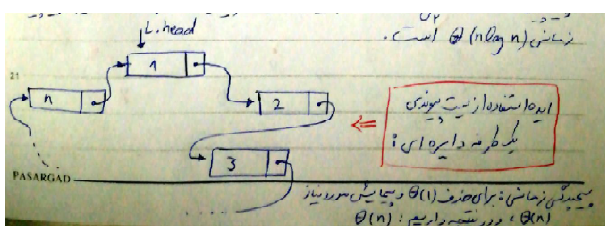
- **گام ۱:** $(n = 9)_{10} = (1001)_2$
- **گام ۲:** $(0011)_2 = (3)_{10}$

حل با استفاده از داده ساختارهای لیست و لیست پیوندی یک طرفه:



- در شروع مقدار flag برای تمام عناصر صفر است.
- هر عنصری که حذف می‌شود، $flag$ آن به ۱ تغییر می‌کند.

پیچیدگی زمانی: $k = \log_2 n$ بار می‌بایست لیست را پیمایش کنیم $O(n)$، در نتیجه پیچیدگی زمانی $\theta(n \log_2 n)$است.



پیچیدگی زمانی : برای حذف $\theta(1)$ و پیمایش مورد نیاز$\theta(n)$، و در نتیجه داریم: $\theta(n)$.